MRC/COM-R-01-945

Mission Research Corporation

PHASE II SBIR FINAL REPORT

DEVELOPMENT OF AN URBAN WARFARE VIRTUAL ENVIRONMENT

31 AUGUST 2001

R. D. EISLER, G. TYRA, AND A. K. CHATTERJEE
MISSION RESEARCH CORPORATION (MRC)
LAGUNA HILLS, CALIFORNIA

MARINE CORPS LOGISTICS BASE
ALBANY, GEORGIA
CONTRACT M67004-97-C-0015 (CDRL A003)

JAMES GROSSE (COR)
SIMULATION, TRAINING, & INSTRUMENTATION
COMMAND (STRICOM)
ORLANDO, FLORIDA

MRC/COM-R-01-945

PHASE II SBIR FINAL REPORT

DEVELOPMENT OF AN URBAN WARFARE VIRTUAL ENVIRONMENT

31 AUGUST 2001

R. D. EISLER, G. TYRA, AND A. K. CHATTERJEE
MISSION RESEARCH CORPORATION (MRC)
LAGUNA HILLS, CALIFORNIA

MARINE CORPS LOGISTICS BASE
ALBANY, GEORGIA
CONTRACT M67004-97-C-0015 (CDRL A003)

JAMES GROSSE (COR)
SIMULATION, TRAINING, & INSTRUMENTATION
COMMAND (STRICOM)
ORLANDO, FLORIDA

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704.0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	31 Aug. 2001	**FINAL REPORT: 1 MARCH 1998 – 31 JULY 2001**

4. TITLE AND SUBTITLE
PHASE II SBIR FINAL REPORT: DEVELOPMENT OF AN URBAN WARFARE VIRTUAL ENVIRONMENT

5. FUNDING NUMBERS

6. AUTHOR(s)
R. D. EISLER, G. TYRA, A. K. CHATTERJEE, AND G. H. BURGHART

M67004-97-0015, CDRL A003

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
MISSION RESEARCH CORPORATION
23052 ALCADE DRIVE, SUITE A
LAGUNA HILLS, CA 92653-1327

8. PERFORMING ORGANIZATION REPORT NUMBER
MRC-COM-R-01-0945

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. ARMY SIMULATION, INSTRUMENTATION AND TRAINING COMMAND (STRICOM);12350 RESEARCH PARKWAY; ATTENTION: AMSTI-ET (JAMES GROSSE); Orlando, Florida 32826-3276

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This effort provides a 3D real-time simulated urban warfare combat environment that allows training and analysis of the actions that take place in that environment without the expense and hazards associated with a live fire exercise. In traditional VR simulations the occurrence of weapon effects does not actually modify the three-dimensional database from which the simulated environment is generated. As a result, dynamic changes to the environment from weapon effects and the interaction of characters in traditional VR simulations must be scripted where the range of outcomes and environmental changes are established *a priori*. Our approach has been to create a simulation that includes the dynamic changes produced by weapon effects on the underlying 3D database. Database changes are governed by sub-real-time algorithms that describe the physics of the respective interaction. Outcomes are not scripted so there is a complete unconstrained free-play of characters with the combat environment. In addition, due to the unconstrained nature of character interaction, the simulation can also be used as a virtual prototyping tool where notional equipment can be tested. The simulation is set in the second floor of the McKenna Town Hall on the urban warfare training facility at Fort Benning.

(REPORT DEVELOPED UNDER SBIR CONTRACT)

14. SUBJECT TERMS
MOUT **Virtual Environment** **Weapon Effects** **Casualty Analysis** **Virtual Reality**
Building **SBIR Report** **Visualization** **Urban Terrain** **Small Arms**

15. NUMBER OF PAGES
89

16. PRICE CODE

17. Security CLASSIFICATION OF REPORT	18. Security CLASSIFICATION OF THIS PAGE	19. Security CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	**UNCLASSIFIED**	**UNCLASSIFIED**	**UNLIMITED**

ACKNOWLEDGEMENTS

We would like to thank the various STRICOM Contracting Officer Representatives (COR) involved in overseeing our work for the Government at various points in time – Mr. Admiral Piper, Ms. Traci Jones, Mssrs, Charles Pope, Louis Garcia, Paul Daumier, and James Grosse. In particular we would like to thank our last COR, Mr. James Grosse, for his patience and understanding with some incredibly frustrating development issues.

We would also like to acknowledge the support of Coryphaeus Software, now part of Centric Software (San Jose, California), which gave use royalty-free use of their Designer's Workbench™ (DWB) software package. DWB was used as the visualization engine for this project. We would also like to thank the Motion Factory (recently purchased by Softimage Co., located in Montreal, Canada) for use of their Motivate™ software. Motivate was used to animate the first generation character simulator incorporated into the STRICOM virtual environment.

Subcontractors on this effort included SA Technologies (SAT, located in Poway, California), GB Laboratories, Inc. (GBL, located in Fountain Valley, California), and Simulation Technologies, Inc. (STI, located in Dayton, Ohio). Mr. Gerry Tyra of SAT was responsible for developing the overall software architecture, visualization system, and software integration that is still on going. Mr. George Burghart of GBL was responsible for the ballistic testing implemented in this program. The experimental data from the ballistic testing was used to validate the sub-real-time physics modules developed by MRC. Mr. Ted Hayes of STI assisted with high-level architecture and distributed interactive computing issues that impacted the software architecture.

Among MRC-Laguna Hills employees, we would like to acknowledge Mr. Mark West (now president of West Engineering, Inc. located in Santa Ana, California), Dr. Amiya Chatterjee, and Dr. Michael El-Raheb. Mr. West was responsible for coding much of the first generation character simulator and developing a test bed for the character simulator interface. Dr. Amiya Chatterjee was responsible for developing software describing the exterior and terminal ballistics of small arm projectiles, fragment distributions from grenades and blast energized windows, and rigid body motion of tables and chairs subject to blast waves. Dr. Michael El-Raheb developed the blast wave model that describes the overpressure evolution in a room from grenade detonation as well as subsequent damage.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

1.1 PROGRAM MOTIVATION

Incorporation of realistic, physically based weapon effects into virtual environments that can be used to train soldiers and law enforcement personnel represents a *holy grail* for the military modeling and simulation community. Currently, visual representations of weapon effects are superimposed on visual simulations almost as an "after thought." As a consequence, physical processes associated with employing a weapon are not incorporated into a simulation. For example, compelling visual representations of the muzzle flash and plume effects from howitzers might be included in a war gaming virtual environment but the effects of cratering, debris, and ejecta (the purpose of firing a howitzer) is not fully evident on the characters or military assets in the simulation.[1] The situation is exacerbated by the fact that current visualization tools do not represent physical properties of objects that might be affected in the simulation, only visible surfaces. So, while photorealistic, visual animations of hypothetical weapon effects might be represented or "scripted" within a simulation, the effect on other objects or non-scripted events become problematic.

The need to realistically simulate weapon effects with sufficient generality and degrees-of-freedom in a training environment has never been more important than in an urban warfare setting where the rules-of-engagement, limits on collateral damage to non-combatants and the urban infrastructure, and weapon performance and effects can vary widely.

Collateral effects of weapons, e.g., fragmentation and back blast, can have disastrous and unexpected consequences in these environments where target engagement ranges can be very close. Urban terrain also provides tremendous variation in terms of weapon effects. For example, the same weapon employed against two different wall constructions can have completely different effects – ricochet versus penetration. 5.56-mm ammunition striking plate glass windows can fragment, ricochet, or be deflected during perforation as a function of striking obliquity.[2] Engagement ranges and times are short resulting in a fast and confusing pace of battle where the risk of friendly fire, ricochets, fratricide, and injury to non-combatants is high.

Addressing these issues entails a high degree of situational awareness and mission planning. This can only be realized by mission rehearsal for all of the unexpected contingencies that an urban warfare environment can provide. Further, these contingencies, particularly as related to weapon effects, cannot be safely employed in a MOUT training course; rather, the only option to safely demonstrate collateral weapon effects is in a virtual training environment.

A key theme of the this effort was to not only represent physical processes engendered by weapon effects where "...everything goes according to plan" and deterministic models are

[1] Example provided by D. Reiss, U.S. Army Infantry School, personal communication, 23 April 1996.
[2] S.C. Robertson, "Rifle Ammunition Performance through Barriers," INTERNATIONAL WOUND BALLISTICS ASSOCIATION REVIEW, 2(4), 1996.

sufficient to describe outcomes but also to incorporate analytical models in sufficient generality, with sufficient degrees-of-freedom, and sufficient detail where things that can and do go wrong are also modeled and represented in the simulation. For example, a small arms model was programmed which includes deterministic and stochastic modules. The deterministic small arms model developed for this project describes the exterior and terminal ballistics of rifle, handgun, and shotgun ammunition striking various types of interior building constructions.

For rifle and handgun ammunition the non-linear trajectory through air, including crosswind effects (which can be very significant in urban environments where wind is channeled), air drag, and gravity as well as striking velocity, obliquity, and hit location on the target are described. Projectile striking conditions are then used in conjunction with a library of projectiles (single slug, and two buckshot loads for a 12 gauge shot gun, 7.62 and 5.56 Full Metal Jacket – FMJ – rifle and 9-mm parabellum handgun ammunition) and a library of interior building constructions (window, wall, ceiling, floor, and interior door cross sections) to determine subsequent trajectory after interaction with these targets (i.e., ricochet, penetration, or perforation). For shot gun loads with buckshot, in addition to the variables described above, shot dispersion as a function of range is calculated.

The stochastic portion of the model, which can be toggled "on/off" with various options, uses the deterministic target hit point calculation to describe the center of a random distribution of target striking locations. A Circular Error Probability (CEP) describes the distribution about this center. The CEP is obtained from experimental databases that tabulate delivery errors in terms of shot dispersion and offset from an aim point. These databases, which are specific to each weapon and projectile, have been collected by the Army to determine the effect of target exposure time, training, stress, and firing conditions (first shot, single automatic shot, burst mode) on gunner accuracy. The external ballistics, deterministic portion of the model has been correlated with data supplied by the Army and ammunition. The terminal ballistics portion of the deterministic model reflects MRC ballistic data generated during the experimental portion of this effort.

1.2 FINAL PRODUCT

The scope of the Phase II virtual environment is shown in Table 1 below. A top-level schematic of the software architecture is shown in Figure 1 (page 14). The final product of this program is a real-time simulation of a building clearing operation on the second floor of the McKenna MOUT Site Town Hall at Fort Benning. The deliverable is a two-player man-in-the-loop system. The only limitation on the number of players however is the number of SGI platforms or Heads-Up-Displays that is available. The STRICOM software will be HLA compatible. There is no requirement to implement the DMSO HLA testing on the simulation software.

The basic difference between the STRICOM/MRC simulation software developed in this program and others is the internal logic of how the STRICOM/MRC system works. There are many visual simulations involving buildings and weapon effects. These simulations tend to share two major shortcomings however. First, the software for these simulations is not governed by the underlying physics describing the processes being simulated. They are strictly visual simulations. Although the visual simulation may be compelling, the physics underlying the processes being

simulated must be scripted. A consequence of this, in the extreme case, is, if a bullet is fired in one of these simulations, you might see a round hole in the immediate wall struck by the bullet as well as a rectilinear array of bullet holes perfectly aligned through an infinite number of walls.

TABLE 1. PROJECT SCOPE

ENVIRONMENT	FURNITURE
Two Player	Steel frame chair
Served by unmodified SGI computer	Wood frame chair
Limited construction material types	Steel desk
Limited furniture	Wood desk
Limited selection of weapons	Steel table
Limited use of sound	Wood table
WALLS	**WEAPONS**
Wood stud and dry wall	M16A1, 5.56- mm N193 Ball
Steel stud and dry wall	M16A2, 5.56-mm M855 Ball
Concrete block	M249, 5.56-mm M855 Ball
Reinforced concrete	M60, 7.62-mm M80 Ball
DOORS	M9, 9-mm M822 Ball
Hollow core wood	12-Gauge Shotgun
Solid core wood	Slugger
Hollow core steel	00 Shot
WINDOWS	04 Shot
Single pane glass	M203, 40-mm, M406 HE grenade
Multiple pane glass	
Plate glass	
Tempered glass	

This leads to the second shortcoming. Since existing simulations are visual, they tend to have decision points associated with scripted outcomes similar to a videodisk system. In some cases these scripts and decision points are sophisticated, but they do not allow for true multiplayer interactively since the range of outcomes are predetermined.

In the STRICOM/MRC simulation, sub-real-time models describing the physics governing the processes invoked in the simulation govern what happens during the simulation. In the STRICOM/MRC simulation there are no decision points or scripts and no outcomes known *apriori*; it is a complete free play of men-in-the-loop representing the characters. Collateral damage can occur and the entire range of mistakes that can occur in this type of urban warfare setting can be represented. For example a grenade can detonate and the fragments can penetrate into the next room depending on the wall construction; bullets can ricochet, wind from channeled air through a broken window can influence the projectile trajectory, etc. When personnel are struck, a detailed penetration analysis is conducted on the internal anatomy of the casualty to determine tissues affected and the extent of wounds.

The simulation begins with the geometry of the building. These geometry files were obtained from the Battle Lab at Fort Benning. The MOUT site is made of cinder-block construction without windows. Since this is not realistic, we have created a library of different walls, doors, floors, ceilings, and windows. The user initializes the model by using the Fort Benning geometry

but prescribing the types of walls, window, furniture, etc. that are in the building. Text character strings in auxiliary files that are read simultaneously with the geometry files represent the construction details. Similarly, the user can prescribe body proportions of the characters and the internal anatomy is scaled accordingly. As part of a parallel U.S. Army Soldier System Command (SSCOM) contract on *Development of a Character Simulator for Battlefield Virtual Environments* we will be able to add protective equipment on the soldier and even notional equipment to see what the effect is during the simulation. Another aspect of the simulation initialization is to decide how the user wants to arm the soldier. A range of small arms is modeled as well as grenades and other types of explosives.

After the initialization, the simulation can begin. When a bullet is fired, the trajectory is calculated and what it hits is determined. In one example, where a 5.56-mm NATO round is fired 45-meters from the outside of the building to the inside of the building and where interior walls are made from gypsum board on 2x4 wood framing, the bullet penetrates all interior walls and exits the building. This required the assessment of more than 1700 graphical constructs to determine if a collision occurred. Once the collision analysis has been initiated, the internal ballistics are determined and if the bullet hits a wall, for example, the determination is made whether the bullet ricochets and what it subsequently hits. If the bullet partially penetrates an object then the damage is determined. If the bullet penetrates the wall then its exit velocity and direction is determined until something else is struck. If a person is struck then the bullet is tracked through the internal anatomy to determine the type of injury. If it exits the person the residual velocity of the bullet is determined.

They key issue on this project was not to create the visual model but how to get the physics to govern the evoked simulation process. As such, visualization development was minimized and several Commercial Off-The-Self (COTS) packages were used in the simulation. SGI's PERFORMER™ software and Centric Software's Designer WorkBench™ (DWB) was used to create the 3D visual environment.

We are using our MRCMAN software developed under another completed SSCOM SBIR contract as the *first-generation* character simulator in the STRICOM simulation. MRCMAN divides the human anatomy into 80,000 voxels and 256 tissue types. The internal anatomy can be scaled for different body types in a linearly proportional manner. It is used for detailed wound ballistics assessments and as the basis of several surgical simulators MRC has been involved with requiring battlefield trauma models. The *second-generation* character simulator is based on the National Library of Medicine's Visible Human anatomy database and will divide the human anatomy into more than 1500 tissue types. The *second-generation* character simulator is being developed in a parallel SSCOM/MRC Phase II SBIR. We have worked with several medical schools on non-proportional scaling of the internal anatomy for different body types and the scaling is based on dissection of more than 2000 cadavers. The MRCMAN software is being animated using the Motion Factory's MOTIVATE™ software. PERFORMER and DWB work on SGI platforms and MOTIVATE requires an NT server.

The fidelity of the weapon's effects software is based on several sources. For the blast calculations we used the U.S Army Waterways Experimental Station's BLAST-X software. This software has empirically based models of internal blasts in buildings. We have correlated our

blast models with BLAST-X and have no more than a 10 of 15% deviation in the predicted internal pressures.

The ballistic simulation was validated by correlating with results from ballistic experiments sponsored by this contract. For every type of wall, floor, ceiling window, and door construction included in our simulation; we have built full scale versions of these targets and used various small arms and measured debris patterns, damage, striking velocity and exit velocity as well as ricochet angles. For shotguns we measured shot patterns on targets at different ranges. The exterior ballistics (projectile trajectory related data) of the small arms data was validated using Government databases, primarily Joint Munitions Effectiveness Manuals (JMEM).

The wound ballistics software MRCMAN is the product of millions of dollars of R&D invested by DARPA and the SSCOM and has been used for casualty assessments by the U.S. Army AMSAA, SSCOM, and in various surgical simulations. As such, it represents the state-of-the-art in conducting casualty assessments for penetrating wounds.

1.3 MILITARY USE

There are at least three military uses of the resulting software: (1) mission rehearsal, (2) training, and (3) virtual prototyping personnel equipment under urban warfare conditions. The U.S. Army Soldier System Command (SSCOM) has awarded MRC another Phase II SBIR to develop an enhanced Character Simulator for the STRICOM virtual environment that will allow for virtual prototyping of personnel protective equipment. Through mutual agreement between STRICOM and SSCOM, the STRICOM software is to be installed at the Defense Simulation Facility at SSCOM in Natick, Massachusetts. A deliverable under the parallel SSCOM SBIR is to replace the first generation character simulator used in the STRICOM simulation with a more advanced character simulator that can more comprehensively address different forms of battlefield trauma. The first generation character simulator, MRCMAN, addresses penetrating wounds only. The second-generation character simulator addresses blast, blunt trauma, and has more advanced tissue penetration algorithms

1.4 COMMERCIALIZATION STRATEGY

There are at two aspects of a potential commercialization strategy for this project. First, is automating the techniques developed in this program to describe weapon effects and making them platform independent so that the weapon effects portion of the STRICOM/MRC software can be imported into other simulation packages. The second, farther term strategy is development of a *Use of Force Simulator* for civilian law-enforcement. There are more than 7 million hours mandated annually in Use of Force training for civilian law enforcement.

Civilian law enforcement represents a very fragmented market however. There are more than 17,000 civilian police departments each of which has different purchasing mechanisms and authorities. Further, within the departments there are very different market segment needs. For

example, SWAT teams need very different attributes in a simulator of this type then patrol officers. Also, the price points for these two market segments are different.

The more general strategy for both the military and commercial users is to automate the techniques we have implemented and make them platform independent so that weapons effects models can be imported to other simulations. Because of the way the visualization system currently works, the weapon effects models will only work with the geometry model in the simulation. From a commercial perspective, it is a demo of what can be done. After going through all of the development cycles however we believe, given the advance in computer technology since this contract was awarded and with video boards in particular, that with minimal additional investment we can generalize and automate many of these techniques as well as make them platform independent.

1.5 PROGRAM PLAN

Three obstacles thwart incorporation of weapon effects on building structural integrity and personnel in real-time simulations. First, most visualization tools merely represent visible geometric surfaces and not mechanical or structural properties necessary to analyze structural integrity or weapon interaction. Second, whereas visualization tools effectively represent rigid-body displacement (translation and rotation) these tools are limited in representing non-rigid body deformation (e.g., crushing, fragmentation, and bending). Finally, the resolution entailed in describing weapon effects and their interaction with objects in the simulation is inconsistent with real-time performance of a high fidelity visual model. Each of these problems is addressed in the Phase II effort.

A phase II library of weapons was developed which include small arms, grenades, and demolition materials. A library of interior and exterior building components is also provided that the user can prescribe in the visual model. Embedded in these building components are parametric solutions for specific weapon-target interactions. An event processor was developed to determine objects affected and the real time of the physical process involved. The event processor invokes and queues algorithms from a "Structural Process" and "Character Process" module that is integrated into the visual system. The Character Process module is a modified version of existing MRC software (MRCMAN) that describes a digital human possessing a virtual anatomy and able to describe trajectories of projectiles penetrating the body. The Phase II software architecture is DIS/HLA compliant (although the appropriate DMSO testing has not been completed to certify HLA status) and permits interface with peripheral devices used in other Army VR applications (e.g., immersive environments). A key theme of the Phase II effort was not only to represent direct effects of the employed weapon but also collateral effects that cannot be safely employed in a non-virtual environment.

Figure 1 shows the overall software architecture of the STRICOM/MRC simulation environment. The grayed areas indicate the focus of the Phase II effort and extensions to the state-of-the art. The non-grayed areas indicate existing off-the-shelf solutions that were exploited and adapted. Three-dimensional visual representations of buildings in general and the McKenna MOUT site in particular are plentiful. However, models suitable for real-time fly-through visual

simulations are not as readily available. A baseline model of the McKenna Town Hall building was developed in Phase I. A more detailed geometry model was obtained from Fort Benning and used as the baseline for the Phase II simulation.

The problems for the simulated effects are threefold in terms of extensions to the state of the art. First, we need to allow "atomic" elements of the simulation that deform in a non-rigid manner (e.g., fragment, crush, bend, etc.). Existing visual simulations cannot do this since they merely render visible surfaces. Second, we need to employ databases with fundamentally different spatial and temporal resolutions to accomplish the above. Further, many of these effects occur in parallel and must be appropriately queued to the number and speed of the hardware/processors being used.

Finally, various decisions must be made. What do we need to show in order to make the simulation compelling. In addition to visuals, what levels of audio or other non-visual cues are necessary? Second, what aspects of the evoked physical processes do not require visual correlates in the simulation but never-the-less need to be known? The "atomic" or fundamental elements of the simulation must also be identified. This last issue involves various trade-offs in terms of simulating rigid-body and non-rigid body processes, the number of polygons in the visual simulation, and real-time performance. Also, what is the appropriate level of approximation and packaging of the physical processes being simulated?

After wrestling with these problems, it became apparent that the weapons effects models cannot be separated from the visualization strategy. The real-time visualization of dynamic changes to the simulation database is the critical constraint. The physical effects models would therefore have to be tailored both in terms of level of approximation and in terms of model parameters so that that a minimum number of atomic elements and polygons occur in the visualization so that the visualization of weapon effects can be done in real-time.

For example, in determining the effect of a shotgun blast through an interior wall, the simulation performs detailed calculations that describe the interaction of the projectiles with each of the wall materials. A library of interior wall constructions is provided that describe projectile interaction specific to the projectile-wall construction combination involved.

For debris fields with casualty producing debris (e.g., glass shards), it is not practical (due to limitations associated with the 3D visualization methodology of debris fields) to track each shard. Rather, a stochastic model was developed that describes the distribution of ejecta and ejecta velocities, and performs a random draw to determine striking conditions on personnel. Once the striking conditions are prescribed, wound calculations (to the level of fidelity required for non-medical purposes) can be done in real-time.

We already have a mature methodology to perform wound analysis on personnel that operates faster than real-time and we merely adapted this for the STRICOM/MRC simulation. In the STRICOM effort we focused on penetrating wounds and production of "operational casualties" as opposed to medical casualties. Casualties are segregated into "superficial" (small diameter soft tissue wounds) and "serious" (penetration into body cavities, bone fracture, severing major blood vessels or nerves) wounds.

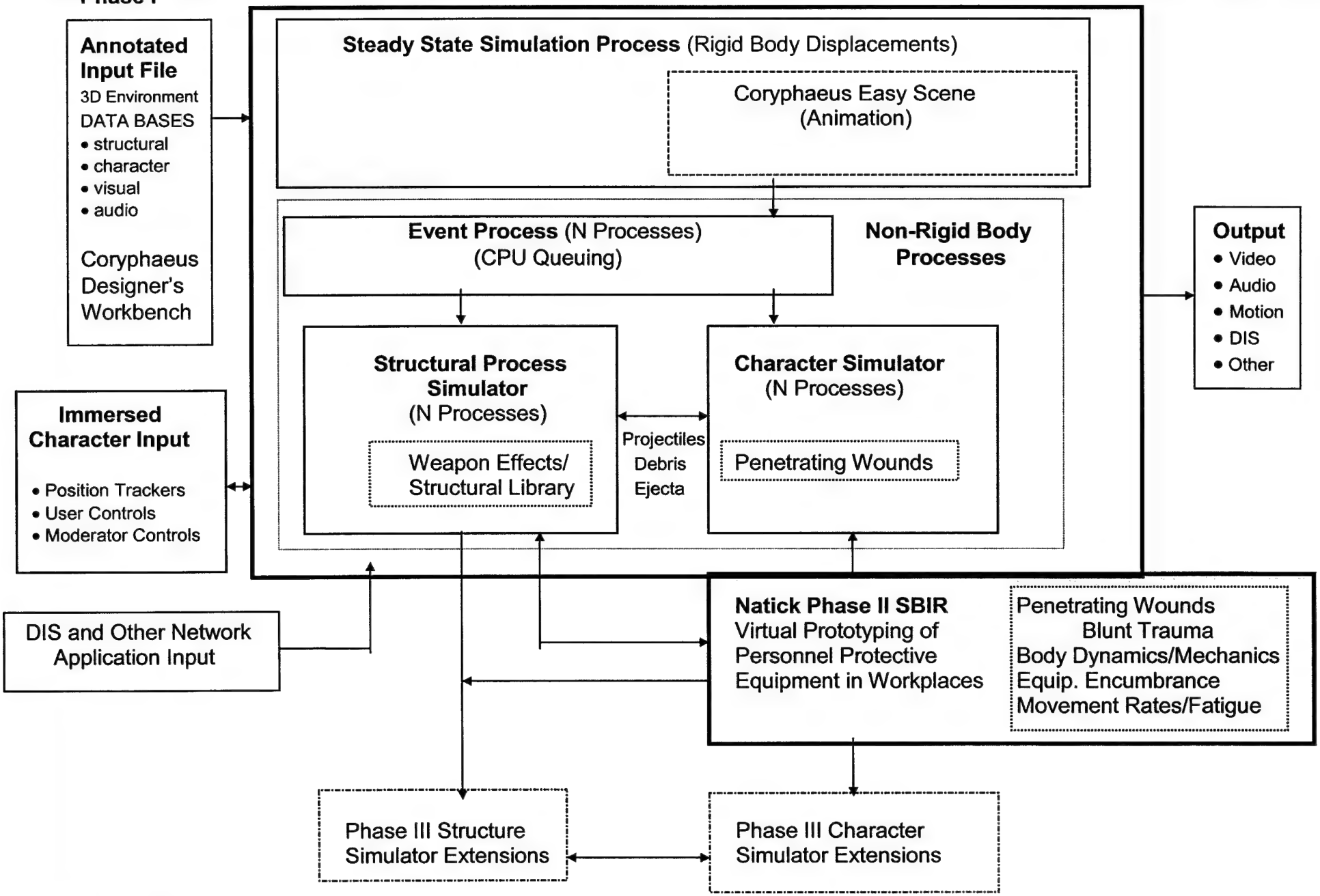


Figure 1. Phase II Software Architecture

In the SSCOM Phase II SBIR alluded to in Figure 1, we focused on other types of trauma which can be exported to the STRICOM simulation. For medical simulation we can provide an interface with third-party surgical simulators so that when someone approaches a casualty and looks down you would switch to the medical simulator where remedial medical procedures could be under-taken under simulated combat conditions.

1.5.1 Software Architecture

Input files are monolithic (i.e. containing both visual and structural information) to maintain synchronization of the data as the number and size of the databases grow. This input format permits interface with other visualization tools.

The overall structure was designed to execute as a single UNIX process. However, in an environment with more than one processor available, various secondary processes can be forked to permit better real time performance. Our hardware environment presupposes multiple SGI Indigos and a SGI High End Impact for development purposes. We are providing an environment with multiple characters interacting (initially two, this is merely limited by the number of processors and input devices available). Two characters will allow us to confront most of the fundamental problems without spending excessive amounts of program resources on hardware.

For any simulation system there has to be a core structure that maintains the real time state of the simulated environment. Using EasyScene™ from Coryphaeus™ Software provided a foundation to build the "Steady State Simulation Process". EasyScene provided real-time video output from an SGI platform. The Steady State Simulation Process controls all of the routine simulation effects that only require translation and rotation of objects (Rigid Body Motion) contained in the database.

An "event" is defined to be any occurrence in the simulation that may physically alter one or more "atomic structures" in the database (Non-Rigid Body Deformation). An atomic structure is defined to be one object or group in the database that has a defined structural composition (e.g. a window or a wall panel).

When the Steady State Process detects that an event has occurred, the event is queued for handling by an "Event Process." There may be more than one Event Process available to handle events, depending on the system resources available. Some Event Processes will require a dedicated CPU to maintain real time performance.

The function of the Event Process is to evaluate the nature of the event and the data base environment in which it occurred. From the center of the event, the environment is evaluated and any atomic objects that may be affected by the event are queued. The interaction of the object and the event not only affects the object, but also may alter subsequent objects that are being checked. This permits shadowing effects for blasts and the determination of penetration effects of firearms.

The Event Process determines those objects which will be affected first (the objects closest to a blast or adjacent objects in the line of fire) and queue the appropriate combinations of "Structural Processes" and "Character Processes." After these processes have evaluated the effect on themselves and the effect of the event, the data is returned to the Event Process. If the event can still affect another object, the next object is queued for evaluation. This continues until the effect of the event dissipates.

The Structural Process has two functions, determine the effect of an event, and create a new visual representation of the object, as it would appear after the event. The visual representation is made up of two parts, the permanent representation and any transient effects (e.g. dust, smoke, debris). The updated visual representation is passed back to the Steady State Process for rendering.

The base requirements for the Character Process are the same as the Structural Process with the addition of character animation and biomechanical functions. These biomechanical functions are emphasized in the SSCOM Phase II SBIR and developed to a much higher level of sophistication and fidelity than in the STRICOM effort. Similarly, the STRICOM effort emphasizes the structural processes to a much higher level of fidelity and sophistication than in the SSCOM effort. Both the STRICOM and SSCOM efforts have a stand-alone level of performance in both the structural and character processes.

1.5.2 Phase II Extensions and Product Development

Future Phase III funding will permit extensions to both the Structural and Character processes to permit a wider range of capability (i.e. more structures, weapon types, higher fidelity biological models, better behavioral models, and other forms of trauma – e.g., diffusion of toxic gases in a structure and effect on enclosed personnel, design of industrial workstations, integration of olfactory sensations and pain stimulus, training devices for civilian law enforcement, etc.).

1.6 TASK DESCRIPTIONS

The Phase II effort consisted of the five technical tasks listed below. In addition to these technical tasks there was an administrative task and one technical task that was deleted and re-directed at the beginning of the program. The scope of each of these tasks will be briefly described in subsections 1.6.1 through 1.6.5 below. Sections 2.0 through 5.0 (Tasks 4 and 5 are combined in Section 5.0) will discuss the software development accomplished in each of these tasks, which are listed below.

Task 1 – Structural Process Simulator Development
Task 2 – Character Process Simulator Development
Task 3 – Ballistic Testing
Task 4 – Software Architecture and Visualization System Development
Task 5 – Network Development

1.6.1 Structural Process Simulator Development

This task is continued from Phase I and consists of developing software and models to address the three issues below.

1. Blast energized fragment distribution
2. Small arms terminal ballistics
3. Blast effects on structural elements including furniture

1.6.1.1 Blast Energized Fragment Distribution. Based on ballistic experiments conducted during both phases of this program, glass fragment production from impinging small arm projectiles will not be a significant casualty producing mechanism in buildings. Glass fragments energized by blast wave impingement however will be a significant factor in causing casualties. A stochastic model describing glass fragment distribution, velocity, and mass was programmed using existing data described in the Phase II proposal. This data gives these quantities as a function of pane area, peak overpressure, window construction, and glass thickness.

1.6.1.2 Small Arms Terminal Ballistics Model. The second subtask concerns validating the Phase I ricochet and penetration algorithms from the ballistic data developed in Task 3 and extending the terminal ballistics algorithms to other building materials.

1.6.1.3 Blast Effects. This task concerns developing a model to describe the spatial distribution and temporal evolution of overpressure inside a building from grenades and demolition materials and the effect on building elements and furniture.

1.6.2 Character Process Simulator Development

The character process simulator uses the MRCMAN software described in the Phase II proposal. MRCMAN divides the human anatomy into 80,000 volume elements and 250 tissue types for the purpose of displaying the trajectory of virtual projectiles through the body. The anatomy can be scaled for different anthropometric body types and articulated positions. Additionally, the projected areas of pre-selected and user prescribed regions on the body can be determined relative to a prescribed plane. In this way, the dynamic probability of different regions being struck by fragmenting munitions can be determined.

The major effort associated with this task is to animate MRCMAN, integrate MRCMAN in a networked environment, and incorporate MRCMAN so that the MRC avatar can interact with the STRICOM/MRC virtual environment. As part of the STRICOM/MRC effort, the first generation character simulator will incorporate MRCMAN. MRCMAN only conducts assessments of penetrating wounds.

When available, as part of the SSCOM Phase II SBIR that is developing an enhanced character simulator for battlefield virtual environments, a second-generation character simulator will be incorporated into the STRICOM/MRC virtual environment. The second-generation character simulator will make operational casualty assessments not only with regard to penetrating wounds, but also blast and blunt trauma. Enhanced penetration algorithms will also be

incorporated into the second-generation simulator. These algorithms will include bone fracture assessments and new coefficients for soft-tissue projectile retardation.

1.6.3 Ballistic Testing

A need exists to define the ballistic, debris, and ejecta environment produced a by a wide variety of projectiles which have penetrated various structural components typical of urban construction. Experimental data is required in three specific areas, namely the visual effects produced by penetrating round, the identification and mapping of ballistically damaging and/or lethal particles produced by these penetrating rounds, including the residual penetrator, and experimental data relative to ricochet.

Dust and fine debris produced by these rounds change light levels and significantly alter visibility. In addition, rounds impacting obliquely on hard surfaces may ricochet. When surfaces like glass are impacted obliquely, the ricocheting round may damage the target and produce fragments from the front surface as well as the rear surface, in addition to the deflected primary projectile.

During the Phase I effort, exploratory experiments were conducted on structural components typical of interior walls in light urban frame construction as well as residential construction. A brief description of these Phase I targets is presented below.

1. Interior wall consisting of two sheets of 3/8-inch "Dry-Wall" or "Sheet Rock" separated by 2"x4" studs.

2. Interior floor/ceiling consisting of carpet, 3/4" plywood, a 10-inch space and 3/8-inch sheet rock. When shot in the opposite direction (sheet rock first) it is called a ceiling.

3. Hollow core interior door consisting of 2 sheets of pressboard, separated by 1.1 inches.

4. Window glass, not tempered.

5. Window/door glass, tempered - safety.

These structures were tested using a 9-mm Parabellum handgun and a 12 gage shot gun employing rounds representative of Police rounds used in urban applications. A FMJ rifle round, 7.62 mm, was used for one test only. All tests were performed at zero obliquity and all weapons were hand-held with rifle and shot guns fired from the shoulder.

For the limited Phase I testing effort, instrumentation was limited to the determination of the residual velocity of the penetrating round, namely the fastest projectile leaving the rear surface, video tape recordings of a nominal 2-foot cubic volume behind the target, and photographs of target surfaces and the fragment spray produced behind the target.

None of the configurations tested produced lethal fragments, with the only real threat coming from residual penetrator(s). This result was expected for all structures tested except the glass.

The videotapes (which were previously sent to STRICOM), in general, showed some slow particles and a significant dust cloud for test involving sheet rock. Two glass panels were tested against #OO Lead Buckshot and neither tempered nor regular window glass produced energetic fragments that would pose a threat to an individual more than 2 feet behind the glass. Also, no particularly sharp or jagged fragments were recovered. In general, the fragments were large and recovered on the ground, within 3-feet of the target location.

The Phase II effort expanded the limited database generated under phase I. Additional structural concepts were identified and representative targets were fabricated. The emphasis was placed on concepts that have greater ballistic resistance than those tested during Phase I. For example solid core doors, fire doors, and metal clad doors fell into this category. These concepts produced more energetic fragments. Additional data was also collected using conditions under which projectiles ricochet. Striking and rebound velocities, incident and rebound angles, and entrance and exit damage for subsequent texture mapping were recorded.

1.6.4 Software Architecture and Visualization System Development

This task consisted of five subtasks: (1) system analysis, (2) development of the Steady State Simulator and Visualization subsystem, (3) Development of the graphic output for the event simulator – i.e., the Structural Process Simulator and the Character Process Simulator, (4) Software test and integration; and (5) Software architecture development and implementation.

In order to meet the technical requirements of the project, the system had to have at least four basic characteristics. The first characteristic is a unified graphics system. There have been a number of simulations available which predict weapons effects. However, what has been missing is a simulation that will allow the effect to be computed and then displayed in a real time training environment. Therefore, the quality of the weapon effect simulation must match the quality of the intended training environment.

Secondly, in order to provide a valid weapon effect simulation, a detailed structural description of the training environment must be provided. Since the training environment must be self-consistent, the data files which describe the environment and its structure must be maintained as a unified whole.

Third, as the complexity of training simulations increases, the computational requirements also increase. In the near term, it is unlikely that a single processor computer environment will be able to provide computational resources that a simulation of this type requires. Therefore, the capability to make use of a multi-processor computer environment is essential.

Finally, a training environment that is useful for teaching urban warfare must have the capability of training a team, not just an individual.

The objective of the System Analysis was to explicitly define the overall system requirements and interfaces, evaluate available off the shelf software components, and determine the functional partition and interface definition of both existing components and new software components. The system interface must take into account the need for the simulation to evaluate

its hardware environment at run-time and launch secondary processes as appropriate. While the multiplayer simulation environment can use a network based communications system; within a single player's environment, communications must be based on shared memory.

The development of the Steady State Simulator, as well as the visual and audio subsystems, is based as much as possible, on commercial, off-the-shelf software. The basic components were built on EasyScene™ from Coryphaeus™ (now Centric) Software. The integration of the structural details was implemented as comments within the graphics databases.

When an event occurs which may result in a non-rigid body change to the database, the event is passed off to the Event Process module. Since the event process has access to both the graphic and structural data for the training environment, it can determine which objects in the environment can be affected. The Structural and Character Processes can then determine how these objects are affected. Once this determination is made, a new set of graphics constructs can be created to represent the new state of the object. These constructs are then passed back to the Steady State Simulator. The Steady State Simulator then deletes the previous display list for the object and creates a new list based on the new state (See Figure 1).

As previously mentioned above, this overall task includes the following items: (1) Conducting system analysis, functional partition, and interface definition for the Steady State Simulator and the Visual/audio subsystem, (2) Development of the Steady State Simulator and the Visual/audio subsystem, (3) Development of the geometry update functions of the Structural Simulator and the Character Simulator, (4) System integration and software testing, and (5) Development of software architecture.

1.6.4.1 Visual Representation of Structural Damage. The approach used to graphically describe structural damage can be described by a simple example. For this example, consider an interior wall panel with wood frame construction. The initial graphic representation would be two single faced rectangles, each of which represents an outer face of the wall panel. If a weapon event causes damage to this wall panel, any given point on a face of the panel will be in one of three states: undamaged, damaged, or missing. To represent the new state of the panel, the center and radius of the area in which there is wall material left will be determined. An imaginary rectangle inscribed by this circle is defined. Similarly, the center and radius are determined which contained the entire damaged area. Another imaginary rectangle is defined which inscribes this circle.

The original vertices of the panel, combined with the vertices defined above, provide all of the required geometric data to build two sets of quadrilaterals. The outer set are undamaged wall elements and have the same color and texture mapping as the original wall panel. The inner set cover that region of the wall, which was damaged. A new, alpha blended texture is created to provide the color, visual texture, and see through properties determined by the appropriate damage simulation. The innermost region has no geometry provided since it no longer exists.

Additional geometry is provided on the back faces of the wall panel to provide for a sense of depth in the wall. However, this geometry would only be implemented in the immediate vicinity of the damage. This is done to eliminate the need to render internal surfaces, which are not

visible. If the damaged wall section includes any studs, geometry and textures will be created to show the studs with the appropriate damage.

1.6.4.2 Visual Representation of Debris and Ejecta Fields. Smoke, particulates and flying debris have a major impact of visibility in a closed environment. These categories can be restated based on the time constants that reflect the behavior of material. Effects with very long time constants tend to be uniform. The fog effects provided by the OpenGL interface was used to represent this.

Fine particulates tend to move slowly and disperse from a non-uniform origin (e.g., a billowing cloud of smoke). This is represented by a series of textured, alpha blended polygons oriented toward the viewer. The density of the particulate field determines the number and visual density of the polygons.

Debris, being more massive, has relatively short time constants. The event happens, debris is blown about, and it settles. Debris is represented by simple pieces of geometry that are given an initial velocity vector and allowed to move and tumble until they strike another surface and/or fall to the ground.

The overall effect would be a timed progression from debris through particulates to fog. As the debris moves through the air, a cloud of particulates would evolve behind it. As the particulates dissipate, the fog level would be increased to some value that is a function of the initial event, volume of the room, and ventilation. Over time, the fog levels would clear.

1.6.5 Network Development

The Under Secretary of Defense has designated high Level Architecture (HLA) as the standard technical architecture for all DoD simulations for Acquisition and Technology. Since compliance with HLA will be required, this effort will follow the requirements of HLA. These requirements direct that each simulation (i.e., a "federate" in HLA terms) and each group of interoperating simulations (i.e., a "federation") follow a set of HLA Rules. They also require development and documentation of HLA Simulation Object Models (SOMs) and an HLA Federation Object Model (FOM).

In the STRICOM/MRC simulation package, interfaces will be developed to the HLA Runtime Infrastructure (RTI) software. A major purpose of the HLA is to provide a common architecture that allows multiple simulations to interoperate without the need to develop specific interface software for each combination of simulations. This promotes simulation/software reuse with minimal effort.

HLA uses the term "federation" to represent the cooperative set of simulations, display systems, loggers, etc., that interoperates for a particular purpose. The HLA refers to each member of a federation as a "federate". Federates cooperate to model entity states and interactions for a particulate domain of interest; an entity in HLA is generically referred to as an "object". An object's state is represented by the values of its "attributes". "Interactions" between federates may cause the attributes of objects to changed. HLA supports the concept of shared and dynamic

ownership for objects attributes across a federation, i.e., different attributes of the same object may be owned by different federates. However, each attribute of each object may be owned by only one federate at a time. Attribute ownership implies the ability and responsibility to update that attribute's value for that object. It also implies the responsibility to "publish" changes to attribute values for other federates that have "subscribed" to that attribute.

A federate can be an active participant in the federation, i.e., the federate owns objects, publishes attribute changes, optionally subscribes to attributes of objects in other federations, and optionally generates or responds to interactions. A federate could also be a passive member of a federation, i.e., the federate only subscribes to the attributes objects owned by other federates.

The HLA rules require the development of a Simulation Object Model (SOM) for each federate participating in a federation and Federation Object Model (FOM) for the federation as a whole. The SOM and FOM must be documented using the HLA Object Model Template (OMT). The use of the OMT promotes the long term and wide usability of a federate by requiring a common descriptive format for the public interactions of that federate. The OMT requires the definition of the relevant information about public object classes, their attributes, and interactions supported by the federate. Support can be in the form of publish, subscribe, or both. The SOM defines the public interactions supported by an individual federate, and can be used to support multiple federations. The FOM defines the interactions between a particular set of federates cooperating to represent a domain of interest.

HLA provides common software to support the interaction of federates in a federation. This software, the HLA Runtime Architecture (RTI), provides a set services required to execute the HLA concept. There are six basic RTI service groups: Federation Management, Declaration Management, Object Management, Ownership Management, Time Management, and Data Distribution Management. Each of these services is accessed through an RTI Application Programmer's Interface (API), that defines the data types, structures, and functions required to interact with the RTI. This effort will require the development of interfaces to the HLA RTI using the API.

In this task an HLA Simulation Object Model (SOM) will be developed for each of the simulations and independent display systems. Also, an HLA Federation Object Model (FOM) for the overall system will be developed. This task is in progress.

SECTION 2.0

STRUCTURAL PROCESS SIMULATOR

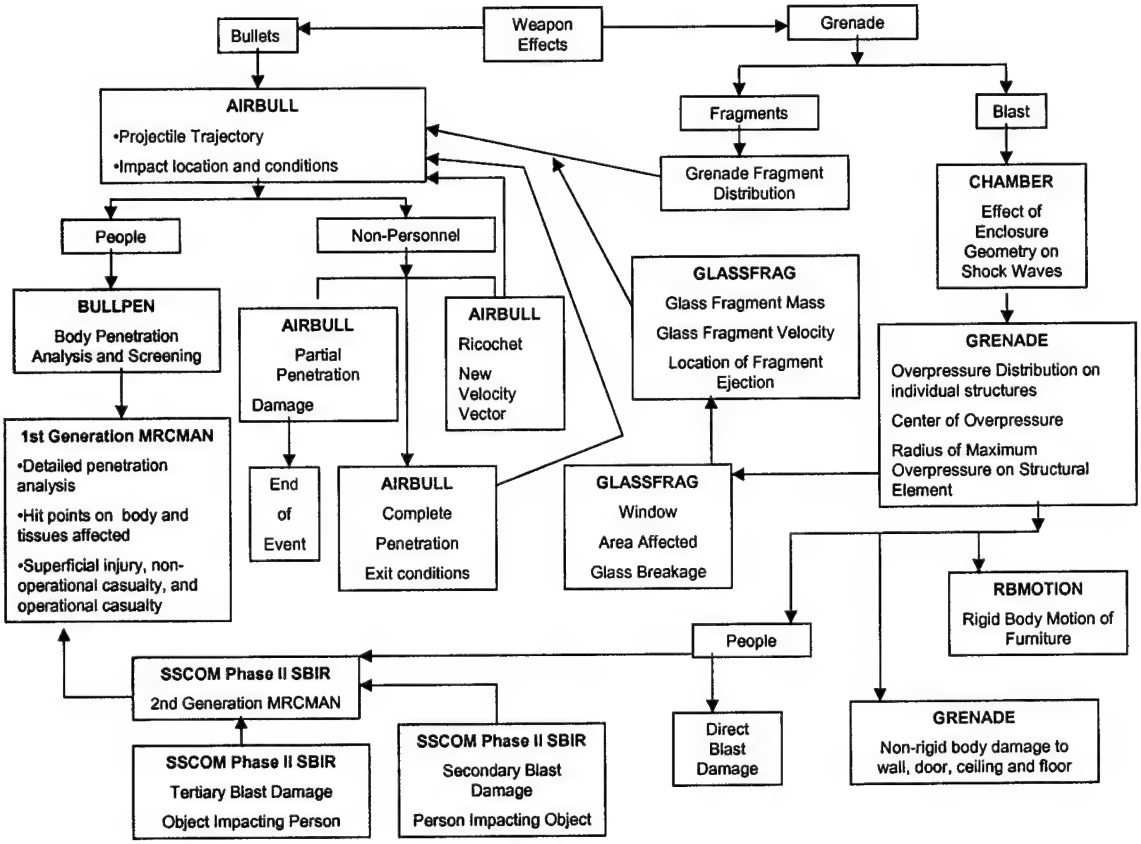


Figure 2. Weapon Effect Modules. Blue filled processes are associated with the Character Simulator development described in Section 3.0. Un-filled boxes are work packages associated with the Structural Process Simulator.

The structural process simulator includes the weapon effects software modules shown in Figure 2 as unfilled boxes. The weapon effects modules developed during this effort can be divided into two categories – projectiles and blast effects (which for the purpose of this discussion also includes blast energized debris). The codes associated with these effects are addressed separately in Sections 2.1 and 2.2, respectively. Section 2.3 discusses the integrated version of the blast related codes and Section 2.4 discusses the codes describing the motion of furniture.

Each code is a compilation of several modules to perform subtasks under a given event as mentioned earlier. Code development was done in three stages. For each subtask, first a MATHCAD (MATHCAD 2000 Professional Version; identified by a .mcd extension in the MCAD code listing in the Section 2.0 Appendix) code is written to check the desired output for a

given input. Since MATHCAD code uses a simple math-like coding structure, these are very easy to check and debug. Second development of the same module is done in FORTRAN, and the input/output are checked by the MATHCAD code. Since the output is already known for a given input from MATHCAD code, each FORTRAN code is revised and debugged until agreement with MCAD code is obtained. In the last step, a C-version of the code was written when required by the final visualization and simulation code for efficient implementation. Each code went through multiple development cycles and the various versions have version numbers as a part of the name and they are updated to new version numbers until all aspects of the problem are included in the code. For example, in the Section 2.0 Appendix, AIRBULL_C_6.cpp indicates version six of the AIRBULL code written in C-language.

2.1 SMALL ARM PROJECTILES

AIRBULL is the main code that deals with the onset of a small arms event and projectile trajectories. The code includes multiple impacts/penetration with other objects and executes until the event stops. This code calculates the projectile trajectory in the air after its initiation. If the projectile hits an object, it calculates whether it penetrates partially, completely or rebounds. If it penetrates partially, the event ends there. Otherwise, the code calculates the subsequent trajectory of the projectile. For motion in space after launch, the code calculates its trajectory using both linear and nonlinear equations of motion using the Lin_Ana and Non_Lin_Ana modules, and assumes that the projectile is aligned with direction of the velocity in the same way it is launched. AIRBULL selects which module to use based on the target range. If the motion continues after ricocheting, then the projectile spinning as well as the translatory phase of the motion are calculated by the BULRICO.for code (see Section 2.2.1). BULRICO.for is a stand-alone code that the bul_ricochet module calls when ricocheting occurs. When penetration occurs, the amount of kinetic energy at impact has to exceed some critical penetration energy. A check on the satisfaction of this condition is done by the Pen_energy module of the code.

The interactions of projectiles with elements of the simulation are handled in the AIRBULL code described below in Section 2.1.1. The original AIRBULL code was written in both C and FORTRAN. This code calculate all the relevant data associated with a complete small arms event starting from the firing of the bullet to the impact and penetration of targets as well as post impact events such as ricochet and collateral effects on other targets. Other than data processing, the main modules of this code that deal with the physics of the flight and impact of the bullet with targets are as follows:

1. Linear version of the projectile trajectory module
2. Nonlinear version of the projectile trajectory module
3. Projectile impact with target
4. Impact velocity and impact location
5. Post impact projectile-target response

For visual simulation of multiple events in a scenario that includes surrounding buildings, furniture and people, it is required to modify the above modules so that they can be used as links to various events surrounding it. Due to these requirements, the input and output to the code are grouped so that a structured input will result in a structured output that will be used as a

structured input to another module. Basically, the new form of the modules allows the users to implement the modules as separate stand-alone codes that do not use any global variables. The usage of global variables may lead to errors when multiple events are processed at the same time by a given module. Out of the above five main modules, the first three have been modified to incorporate the required grouped or structured input and output.

Two sets of structured inputs have been used to pass data to these modules. These inputs are listed below.

Set 1. The members of this structured input are:

1. init_vel (initial velocity of the projectile)
2. elevation (elevation of the muzzle velocity with the horizontal)
3. wind_vel_X (horizontal component of the wind velocity)
4. wind_vel_Y (vertical component of the wind velocity)
5. A, N (Ballistic coefficients of the projectiles)
6. critical_range (horizontal range for the calculation of the projectile path)

Set 2. impact_data structure

1. proj_id (to identify the projectile in the database)
2. target_id (to identify the target in the database)
3. inc_vel (incident velocity for target impact)
4. imp_ang (impact angle for target impact)

The ballistic_input structure enters both the linear and nonlinear modules as structured input, and these modules output the location of the terminal point of a segmented ballistic event in a user-defined frame. The impact data structure enters the target impact module as a structured input, and this module outputs the exit/rebound velocity of the projectile. These data is then passed on to the linear and/or nonlinear modules for processing of subsequent events.

2.1.1 AIRBULL module

Final Version: 8
Development Language: C, FORTRAN and MCAD (some modules)
Number of Modules: 23

A brief description and names of modules

get_sys - User Coordinate system

print_sys – Print current system

print_coord_sys – Print use system

save_his – Saves history file of projectile motion

bul_info – Reads bullet description

bul_drag_coef – Calculates the air-drag coefficient for bullet motion

Lin_Ana – Calculate trajectory using a linearized theory

Non_Lin_Ana – Calculates trajectory using a nonlinear theory

y1dot2 – Defines the differential equation of motion for the horizontal direction

y2dot2 – Defines the differential equation of motion for the vertical direction

clrscr – clears the screen for user input

RKNLR2 – Nonlinear Runge-Kutta algorithm for solving equations of motion

Save_data – Saves data for future or other modules at a later time

Plot_files – Creates data files for plotting

Pen-energy – Calculates required penetration energy for penetration

Target_Damage – Calculates the damage in the target

Impact_Dynamics – Determines the post impact scenario

Proj_rebound – Determines projectile kinematics after rebound

Sphere_rebound – Determines spherical projectile kinematics after rebound
Save_data2 – Saves data for future use

Bul_ricochet – Calculates projectile motion after it ricochets. Uses BULRICO.FOR code.

Save_data3 – Saves data for future use

2.1.2 BULRICO module

Only a FORTRAN version of this code has been developed even though the results have been verified by other known results for some specific cases. This code calculates the projectile motion after a ricocheting occurs. The post impact motion of the projectile consists of two parts; one describes the translatory phase of the motion while the other one addresses the rotation or spinning of the projectile. The second phase is not present in problems where the impulsive force due to impact passes through the center of mass of the projectile. Assuming that the impact induced force is normal to the impact surface i.e., ignoring frictional impulsive force, spherical projectiles do not undergo spinning. Both phases of the motion are governed by the differential

equations of motions of the projectiles. These equations of motion are solved by this code when the initial velocity of the center of mass and the angular rate of motion about the center of mass are given. The outputs of this code are the history of the center of mass, angular velocity and angular description of the projectiles with respect to the center of mass.

2.2 BLAST EFFECTS

Two computer programs, based on DOE/TIC-11268, were developed by MRC and compared with results from the U.S. Army Experimental Station's BLAST–X code. The two MRC codes include: (1) "GRENADE" which computes the overpressure and impulse from a spherical explosion near a rigid wall including the effect of ground, and (2) Program "VENTEDBLAST" which computes the overpressure and impulse from a spherical blast enclosed in a rigid rectangular chamber including inertial venting. In both programs, the effect of steel and fragmenting casings of the grenade are considered empirically. The effect of ground proximity is considered as a multiplicative constant. The magnification of overpressure from multiple wall reflection is considered empirically by experimental data as well as a constant factor to overpressure usually assumed to be 1.75.

Decay of the overpressure from the explosion is assumed to drop exponentially with time based on time constants determined from empirical curves referenced in the above DOE manual. Spatial distribution of overpressure along a wall follows the slant length equivalence instead of angle of incidence. In this way overpressure is reduced uniformly from a minimum standoff distance from the wall. Limited experiments at low overpressures (consistent with grenade yields) suggest that up to 40-degree incidence, overpressure is uniformly distributed over the footprint of the blast on the wall. This footprint is the intersection of the spherical wave front with the plane of the wall.

BLAST-X was exercised on geometries that included a single wall with ground effects as well as the case of a cubical measuring 15' x 15' x 10' at a symmetric stand-off of 7.5 feet from each wall and 1 foot from the floor. The results compare satisfactorily with the "Grenade" and "Ventedblast" models developed for the STRICOM/MRC simulation. BLAST-X and the STRICOM/MRC models show little effect from multiple wall reflections in the chamber for yields up to 2 lbs TNT, which is well below the net explosive weight for a grenade.

A computer program "PANELD" was also developed to compute transient response of a rectangular panel simulating a door or window or a span of dry wall between wood frames. The goal was to realistically predict failure of these components from the blast loading of the grenade. The included algorithms are based on the flexural elasto-dynamic equations of Mindlin, which accounts for finite velocity shear wave propagation. The actual loading from the "Ventedblast" simulation is considered which yields an expanding footprint of pressure varying radially. These effects are important in the simulation as standoff distance from the wall is small and detailed pressure over the footprint is significant in describing stress wave propagation producing structural failure.

Finally, a code was written that calculates the worst-case fragment that can hit personnel in a room following an explosion. Once the worse case fragment is determined MRCMAN is

exercised to determine casualty status. The code consists of a bivariate normal distribution of fragment mass and velocity. Location of a potential strike on the body is determined stochastically. This code is used to determine casualties from window fragmentation and grenade fragments. A flow chart for this code in Figure 3 below.

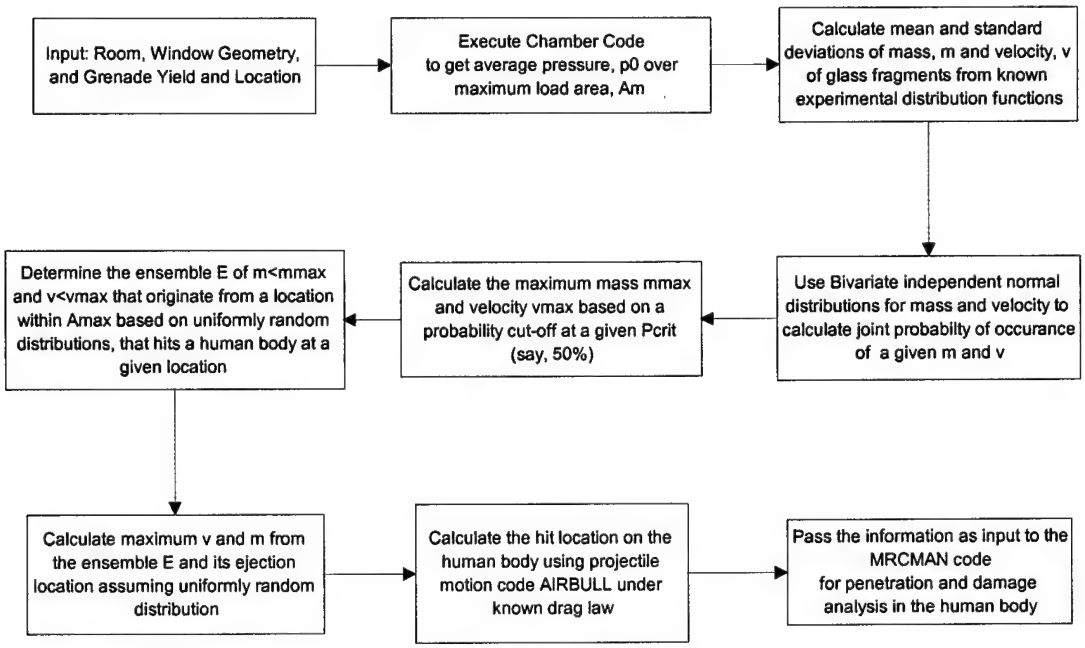


Figure 3. Flow-Chart of how "Worst Case" Fragment is used in Conjunction with MRCMAN Character Simulator.

Currently, the data incorporated into the STRICOM/MRC blast simulation software is contained in Figures 4.5, 4.6, and 4.7 of DOE/TIC-11268. This allows prediction of overpressure peak, time-dependence, impulse, and dynamic pressure measured for a location at a standoff distance that is either at a reflecting surface or else "side-on". The blast may be detonated either in free air or else near a floor. Also the explosive type may one of those given in Appendix A[3] of DOE/TIC-11268 and the weight of explosive may be varied. The distance and weight permit Sach scaling. Currently, the STRICOM/MRC blast simulation software includes: (1) Effects of

[3] Appendix A consists of 9 sets of tables listing various properties of 40 different explosives as follows:

Set 1. Lists common and chemical names, formulation, physical state and specific weight
Set 2. Lists molecular weight, melting point, vapor pressure and toxicity
Set 3. Lists heats of formation, detonation and combustion
Set 4. Lists results of drop weight, skid and friction tests
Set 5. Lists explosion temperature, and results of rifle bullet, Susan and gap tests
Set 6. Lists Hugoniot data and intercept of the unreacted HE
Set 7. Lists coefficients of thermal conductivity and expansion
Set 8. Lists volumetric thermal expansion, specific heat and thermal stability
Set 9. Lists detonation velocity, pressure, TNT equivalent weight and critical energy

detonation height, (2) Effects of casing type and weight, and (3) Effects of an explosion in cubicles, corridors, and bays.

As mentioned above, for the purpose of this discussion, blast effects really consist of two effects in the STRICOM/MRC simulation – the direct effects of the blast wave and the effect due to blast energized debris consisting of glass fragments from windows hit by a blast wave and ejected fragments from a grenade or other demolition material. The individual software modules developed to handle the blast energized debris problems are briefly described in Sections 2.2.1 through 2.2.3 below and then the integrated version of these modules, including direct blast effects, is described in more detail in Section 2.3.

2.2.1 GLASFRAG module

This code calculates the properties of window glass fragments that are ejected due to overpressure due to shock wave blast from a grenade or any other explosion. The location of the window can be specified from the user coordinate system. The code reads the coordinates of a corner of a window so that the height and width can be specified along the positive direction of the axis-system. The coordinates and the radius of the overpressure region on the wall that contains the window are also inputs to the code. If the overpressure exceeds some threshold value, the window breaks into fragments. This value is estimated from experimental data on window glass breakage from explosive blasts. The code calculates the mass, velocity and location of a typical fragment by using known random distribution functions of these variables. The information is then passed on to the AIRBULL code for continuation of the event. In reality, many such fragments are ejected but a typical fragment is used for graphic visualization of the event simulation.

2.2.2 CHAMBER and GRENADE modules

When an explosion occurs due to a grenade or any other explosive sources, it radiates shock waves. The anisotropy in the radiation pattern of the shock wave exists due to the packaging of explosive chemicals inside a grenade. In our analysis, we have ignored this anisotropy and have modeled the shock from grenade explosion as spherical, isotropic waves. The overpressure distribution on an impinging wall depends on the location of the explosive source with respect to the wall, and also depends on the location of other surrounding walls or physical boundaries. The CHAMBER code handles this aspect of the problem. In this case, a chamber is defined as an enclosed domain in which an explosive event occurs. On any surrounding wall, this code calculates the temporal nature of the overpressure as a function of the location, yield of the explosion and the locations of the other surrounding boundaries. The GRENADE code only considers the effect of a wall and a floor in altering the overpressure distribution on walls.

2.2.3 RBMOTION module

This code calculates both the motion of the center of mass and rotation about the center of mass of a rigid body using the classical rigid body equations of motion. The motion of the center of mass is governed by the vector sum of all forces acting on the body while the rotational motion of the center of mass is governed by the sum of the moments of the loads about the center of mass. Since bullet impact on an object like furniture, is impulsive in nature, we use the impulsive force to calculate the changes in linear and angular velocities right after impact, and then use them as the initial conditions for solution of the rigid body motion. During this part of the simulation, we assume that there is no other force acting on the body so that both angular and linear velocities are preserved.

2.3 INTEGRATED VERSION OF CHAMBER, GLASS AND GRENADE FRAGMENT MODULES

The code GRENADE is an integrated version of three separately developed independent codes discussed above, CHAMBER, GLASFRAG, and GRENFRAG. The input required by these three codes has also been integrated into a single input data-file that contains the room geometry, grenade location, size, orientation and strength of the grenade, target location; and the location and dimension of a glass window inside a closed room where a grenade of known size and strength explodes. For the development of the GRENFRAG code, we have used the properties of M406, Composition B HE grenades.

2.3.1 CHAMBER Code

CHAMBER is a Fortran language code based on Fortran 90/95 and calculates the overpressure distribution from a grenade explosion inside a vented or closed rectangular chamber or room. The temporal nature of the overpressure distribution in space or any surrounding wall is based on a series of experimental, overpressure distribution curves from free field grenade explosions. These curves have been documented in tables 4.43, 4.44, 4.45, 4.49, 4.50, 4.51, 4.52 of a DTIC document titled "A Manual for the Prediction of Blast and Fragment Load on Structures", (DOE/TIC-11268, November 1980).

The CHAMBER code identifies the boundaries of a room as front wall, back wall, left wall, right wall, floor and ceiling. These identifications are used with respect to an observer who stands inside the room. The wall in front of the observer is the front wall, the wall on the left is the left wall, the boundary above the observer is the ceiling etc. Wall numbering also starts with the front wall as one, and sequences with left wall, back wall, right wall, ceiling and floor. For each boundary, CHAMBER calculates the temporal distribution of the overpressure. By integrating the overpressure over the footprint of the blast wave front, the CHAMBER software module calculates the maximum force on the wall and the corresponding footprint radius at any given

time. In the integrated GRENADE code, the front wall is programmed to be the wall that contains the window. The overpressure and the footprint radius on the window wall corresponds to the time when the force on the window wall is maximum. This information is then passed on to the GLASFRAG code for the calculation of any window glass fragments that may be ejected from the blast overpressure.

2.3.2 GLASFRAG Code

GLASFRAG is also a code written in Fortran 90/95 language. This code calculates the characteristics of window glass fragments due to explosion-induced overpressure. The statistical distribution of mass and velocity of glass fragments due to shock overpressure have been determined by using experimental data from field explosions. Most of these experiments use far field shock data that is uniformly distributed over the glass surface of a window. For applications in this code, we use the mean mass and velocity as functions of the peak overpressure, p. The location of the fragment origin is determined by using a uniformly random distribution of the fragment location over the exposed region on the glass surface. For grenade explosions inside a room, the regions of overpressure on a given wall grow as the circular shock wave front extends over the wall. As the wave front grows with time, its pressure intensity diminishes while the area covered increases. Thus the overall force exerted on the wall starts from zero, then grows to a maximum prior to diminishing to zero again. For the calculation of the peak overpressure on the wall, we use the average pressure corresponding to the maximum force on the wall where the window is situated. Depending on the location and dimensions of a given window on a given wall in a room, the footprint of the circular wave front corresponding to maximum force on the wall, may or may not completely contain the full window. In the event that this footprint completely covers the window, we have a relatively simple case of determining the fragment ejection location on the window. For partial coverage of the window by the shock wave footprint, we need to determine the exposed area prior to the determination of the fragment ejection location.

In reality, fragments of all sizes and masses are ejected when a glass window is shattered due to blast induced shock waves. For a uniformly distributed pressure field, the distribution of the sizes (and hence masses) and velocities as functions of the peak overpressure has been determined from experimental data. The mean values of these two critical parameters of glass fragments are also functions of the peak overpressure. The GLASFRAG code calculates the property of one such fragment that is ejected from a location based on uniform random distribution over the exposed area. The fragment size is calculated from the mean mass and ejection velocity is calculated from the mean velocity. The mean size (mass) and the mean velocity are calculated from the experimental functional dependency of these quantities on the peak pressure. As mentioned before, the peak pressure is calculated from the average pressure when the force on the wall window is maximum. The mass and size of the glass fragment is then passed on to the MRCMAN code for injury assessment of personnel in the vicinity of the explosion. MRCMAN is a code developed by MRC that describes the path of a fragment inside a human body.

2.3.3 GRENFRAG Code

The integrated version of the GRENADE code also contains the determination of the mass and velocity from the explosion of M406 HE, composition B grenades. A typical distribution of mass and velocity of steel fragments from these grenades are shown in Figures 4 and 5. A sketch of the grenade location and polar angle is shown in Figure 6. The data used in Figures 4 and 5 are obtained from Chapter 3, Section 1 on fragmentation, FM 101-62-3, 61S1-3-4, FMFM 4-7H-3, OP 4215.

Coordinate Systems

There are two coordinate systems used in the integrated version of GRENADE. One is called the user frame and the other is called the window frame. Descriptions of these two frames are given in Figure 7 (page 36).

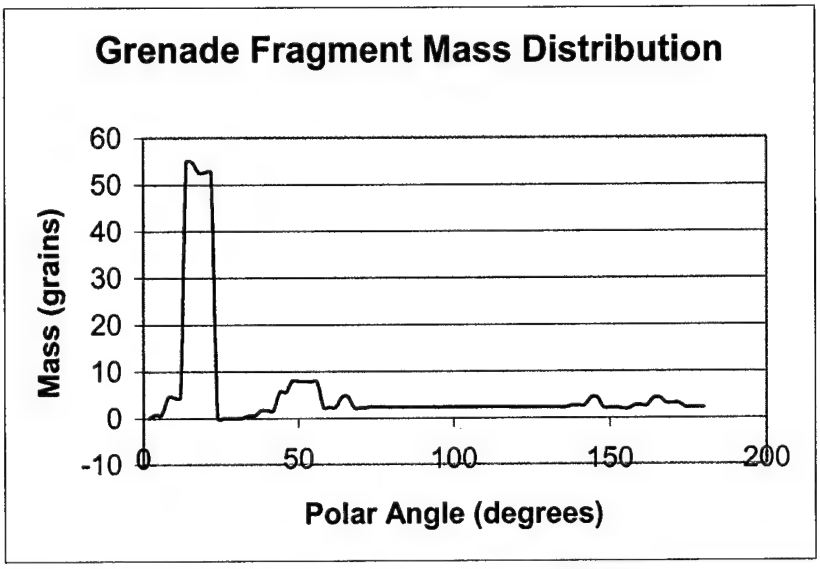


Figure 4. Distribution of Steel Fragment Masses from M406, Comp. B HE Grenades

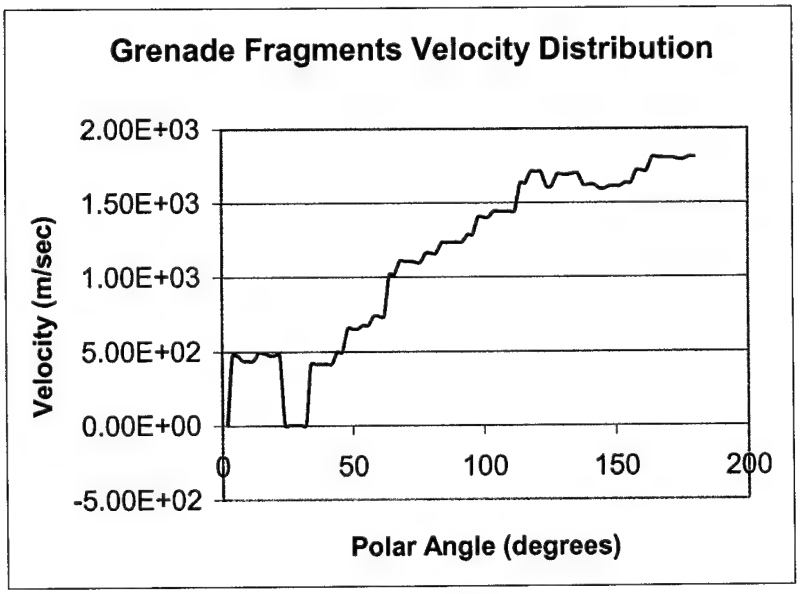


Figure 5. Distribution of Steel Fragment Velocities from M406, Comp. B HE Grenades

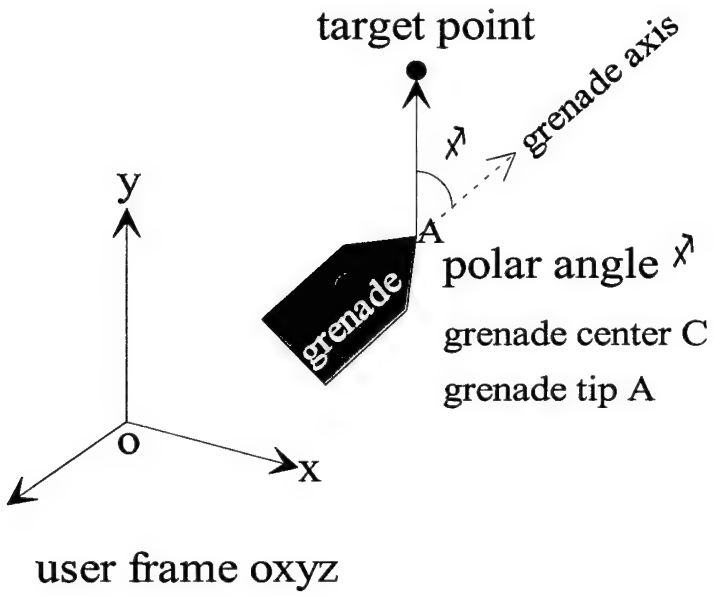


Figure 6. Schematic of Grenade Location in the User Frame and Polar Angle

User Frame

The origin of the user frame is at one corner of the room O, the y-axis is vertical, the z-axis is along the depth of the room while the x-axis is along the width of the room. The definitions of width and depth are arbitrary and should be interpreted as the room dimension along the x- and z-axis, respectively.

Window Frame

In the window frame, the origin remains the same as the origin of the user frame. The y-axis is the vertical axis, the x-axis is aligned with the direction of the line joining the lower left hand corner of the window with respect to an observer inside the room and facing the window and the right hand corner of the room. The z-axis is the inward normal from the window. The descriptions of these axes system for the four possible locations of the window are shown in Figure 7.

Input/Output of GRENDE Code

Input File: *grenade.in*

Line	*Input Parameter*	*Description*	*Unit*
1	RH, RW, RD	Room height, Room width, Room Depth	feet
2	GX, GY, GZ	Grenade Coordinate in user frame	feet
3	WX, WY, WZ	Window Center Coordinates (user frame)	feet
4	WH, WD	Window Height and Length	feet
5	EXP_NO	Explosive Number	Number
5	EXP_WT	Explosive Weight	Pounds
5	VENT_AR	Vent Area	Sq. Feet
6	gtx, gty, gtz	Location of Grenade Tip (user frame)	feet
7	tx, ty, tz	Location of the Target Point (user frame)	feet

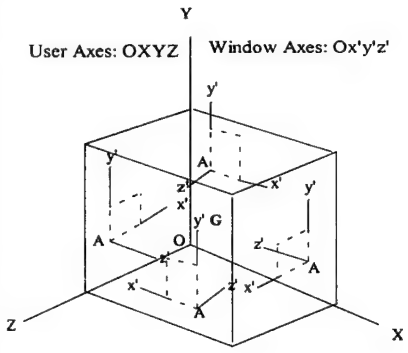
Output Files

Three output files are written every time GRENADE is executed. These files are grenade.out, grn2.store1 and grn2.store2. The grenade.out file contains data relevant for checking input data and output data for its later integration with the visualization code. It starts with the dimension of the room; grenade location, size and strength; window dimensions and location. The temporal distributions of overpressure on all room boundaries are included in this output file. The corresponding maximum forces, time of occurrences and overpressure on all boundaries are shown next. The output from CHAMBER code terminates here and is followed by the output from the GRENGLAS code. It gives the window location, exposed area, overpressure and the calculated glass mass, velocity and ejection location. The last entries in this data-file are the grenade fragment properties along with mass and velocity data.

Output data from the GRENADE code that are required for the visual simulation routine are the size, shape, velocities from the window glass fragments and steel fragments from the grenade. Since the data structure used in the visual simulation code is passed on to other routines as data structures (C-format), some input modifications need to be done prior to its integration with the visual simulation code.

A flow chart of the integrated Grenade related codes is shown in Figure 8 on page 37.

GRENADE Code/User and Window Frame Description



User Frame OXYZ with origin at one corner
Y: Vertical along height H
Z: Horizontal along depth D
X: Horizontal along width W
Room occupies $0 \le x \le W, 0 \le y \le H, 0 \le z \le D$
Grenade location: $G = (G_x, G_y, G_z)$ (user frame)

Wall Identification:
Wall-1: $x = 0$ Wall-4: $z = 0$
Wall-2: $z = D$ Ceiling: $y = H$
Wall-3: $x = W$ Floor: $y = 0$

Window Frame $Ox'y'z'$

Facing the wall containing the window:

Lower left corner is A
x' along the window length from A
y' vertically up from A
z' towards the room from A
Origin is at O

Wall identification in the CHAMBER Code:
For an observer inside the room facing a wall,

Front Wall – Wall-1
Left Wall – Wall-2
Back Wall – Wall-3
Right Wall – Wall-4
Ceiling – Wall-5
Floor – Wall-6

Integration in GLENGLAS Code

Front wall is coded to be the wall that contains the window. Thus CHAMBER code overpressure on wall 1 is the overpressure on the window wall. Window fragment data is based on this data from CHAMBER code.

Figure 7. Description of Coordinate Systems in GRENADE Code

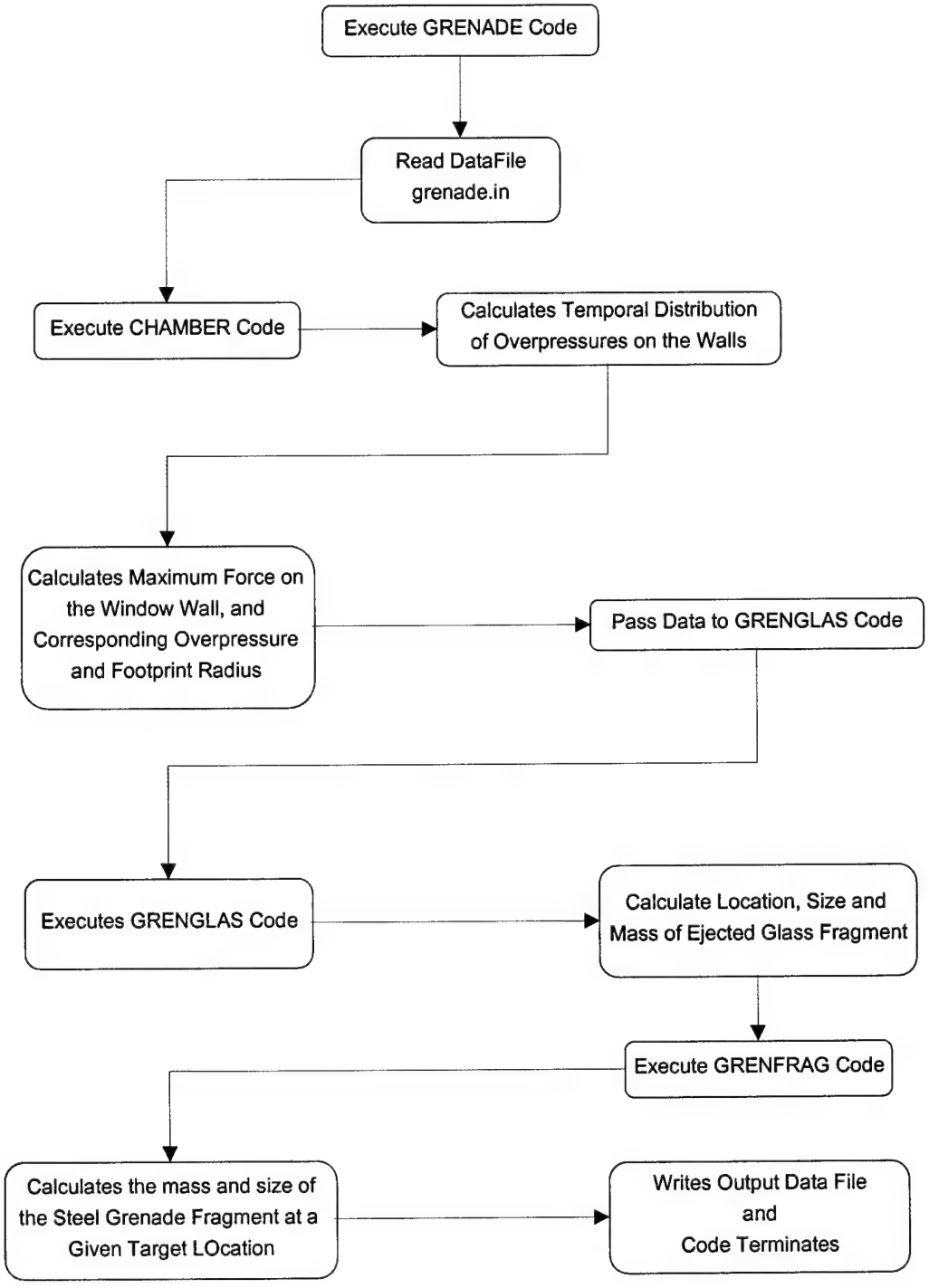


Figure 8. Flowchart of Integrated Grenade Code

2.4 FURNITURE RIGID BODY MOTION

The rigid body motion of chairs (section 2.4.1), desks and tables (section 2.4.2) are described using the codes in this section.

2.4.1 Chairs

A schematic of the STRICOM chair model is shown below in Figure 9.

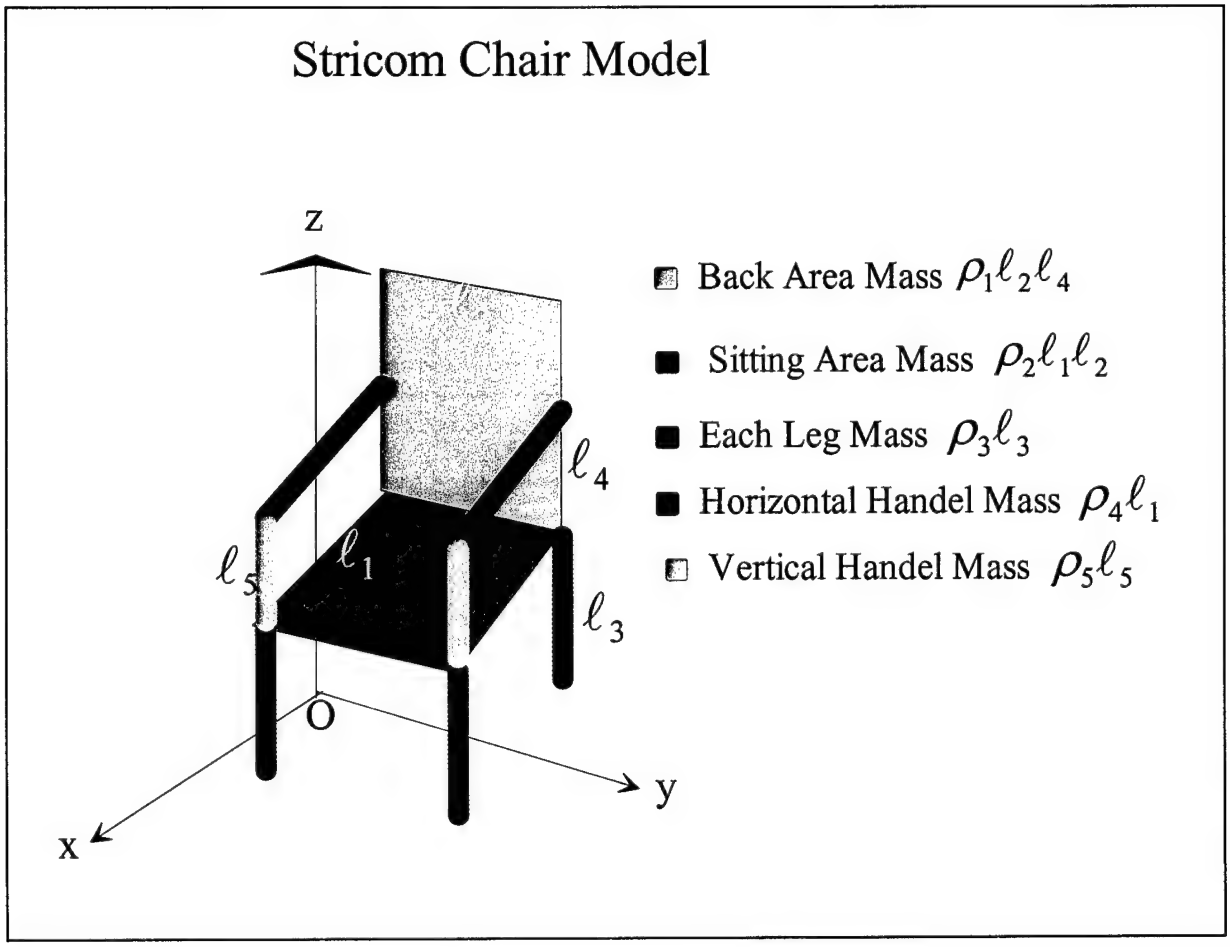


Figure 9. Schematic of Chair Model

The rigid body motion of the chair is determined by using classical, decoupled rigid body equations for the three rotational and three translational modes of the motion. In general these modes are coupled, and the solution involves dealing with nonlinear differential equations of motion since friction is present in the problem. The floor friction is determined by the floor type (carpeted or no carpet). By assuming that these modes are uncoupled, we have introduced some error in our predictions, but since these will be used in the simulation of these modes in our final

visual environment, we feel that these suffices to be first order estimates of these modes of motion. In Figure 9, the densities of various parts of the chair are as follows. ρ_1 and ρ_2 have the dimensions of mass/area while ρ_3, ρ_4 and ρ_5 have the dimensions of mass/length. In our chair model, we used solid oak material for all parts of the chair, with a density of $0.0255\frac{lbm}{in^3}$. These values have been adjusted according to the cross-section of these areas. Other data used for our chair model are shown below.

Data used in the Chair Model

Stricom Chair Model: Material Solid Oak: Density ρ := 0.0255 **lbm/in**3**
All length dimensions are in inches; Mass in lbm; Force in lbf; Impulse in lbf-sec;
Velocity in in/sec or rad/sec

Legs: Density ρ3 := ρ Cross-section A3 := 2·1 Length h3 := 19 Mass m1 := ρ3·h3·A3 Mass = m1 = 0.969

Sitting Area Density ρ2 := ρ Dimension h1 := 20 and h2 := 20 Thicknees hs := 1 Mass m2 := ρ2·h1·h2·hs Mass = m2 = 10.2

Arm Rest Horizontal: Density ρ4 := ρ Cross-section A4 := 1·2 Mass m3 := ρ4·h1·A4 Mass = m3 = 1.02

Arm Rest: Vertical Density ρ5 := ρ Height h5 := 6.0 Mass m4 := ρ5·h5·A4 Mass = m4 = 0.306

Back Area: Density ρ1 := ρ Thickness hb := 1.0 Height h4 := 12.0 Mass m5 := ρ1·h4·h2·hb Mass = m5 = 6.12

Projectile Impact

Translation. Let G be the center of mass of the chair and a projectile of mass m_b impacts at the point A on the chair with a velocity v_b. Then the input momentum is $I_0 = m_b v_b$. If the impact direction at A is given by the direction cosines $\vec{I} = \begin{bmatrix} I_x \\ I_y \\ I_z \end{bmatrix}$, then the linear impulse is given by $\vec{L} = I_0 \vec{I}$. Using the impulse-momentum equation for the ensuing chair motion, initial velocity $\vec{V}_0$ imparted to the chair due to the linear impulse by the projectile, can be calculated from

$$\vec{V}_0 = \frac{\vec{L}}{m} = \begin{bmatrix} V_{0x} \\ V_{0y} \\ V_{0z} \end{bmatrix} \tag{1}$$

As indicated earlier, we assume that the motions of the chair along the axes are uncoupled. To calculate the linear translation of the chair along the x-, y- and z-direction, we use the equation of motion of the chair along these directions. In the presence of friction with a coefficient of kinematic friction μ_k, the equation of motion in the x-direction is given by

$$\ddot{x} = -\mu_k g \tag{1a}$$

Solving this equation with the initial condition $x(0) = 0; \dot{x}(0) = V_{0x}$, we have the distance x_d moved by the chair along the x-direction as $x_d = \frac{V_{0x}^2 \operatorname{sgn}(I_x)}{2\mu_k g}$. Similarly, the distances y_d, z_d moved by the chair along the y- and z-directions are given by $y_d = \frac{V_{0y}^2 \operatorname{sgn}(I_y)}{2\mu_k g}$ and $z_d = \frac{V_{0z}^2 \operatorname{sgn}(I_z)}{2\mu_k g}$, respectively.

Rotational Motion. Impact induced impulsive moment $\vec{M}_G$ about the center of mass G can be calculated by taking the moment of the impulse $\vec{L}$ about G; this gives $\vec{M}_G = \overrightarrow{GA} \times \vec{L}$. Thus the components of initial angular velocity vector ω_{x0}, ω_{y0} and ω_{z0} of the chair about the user axes can be calculated from

$$\omega_{x0} = \frac{M_x}{IG_{xx}}, \omega_{y0} = \frac{M_y}{IG_{yy}}, \omega_{z0} = \frac{M_z}{IG_{zz}} \tag{2}$$

where IG_{xx}, IG_{yy} and IG_{zz} is the moment of inertia of the chair about the $x-$, $y-$ and $z-$axis, respectively.

Figure 10 shows the schematic of chair rotation about the x-axis through an angle θ.

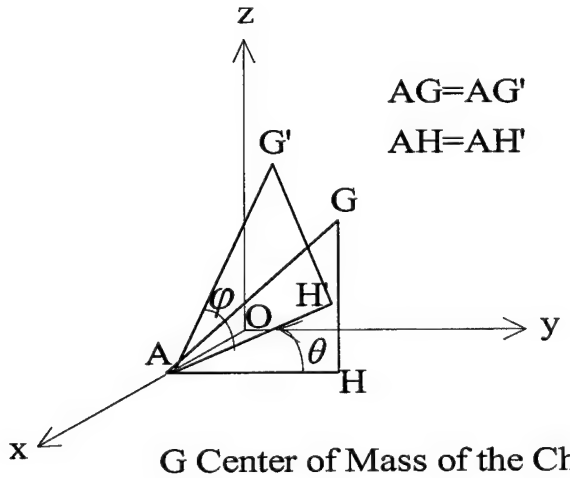


Figure 10. Schematic of chair rotation through an angle θ

to calculate the rotational motion of the chair due to these initial angular velocities, we solve the differential equations of motion that governs this mode of the motion. With respect to Figure 10, these equations are given by

$$\begin{aligned} I_{Ax}\alpha_x &= I_{Gx}\alpha_x + mc_x^2\alpha_x = -mgc_x\cos(\theta_x+\varphi_x) \\ I_{By}\alpha_y &= I_{Gy}\alpha_y + mc_y^2\alpha_y = -mgc_y\cos(\theta_y+\varphi_y) \end{aligned} \tag{3}$$

where the subscripts x, y indicate the axis about which the rotation takes place and the other quantities have similar identities. Subscripts A, B indicate the effective points of rotation of the chair (Figure 10). Integrating (3), we have the general form of the solution as

$$\omega^2 - \omega_0^2 = \frac{2gc}{k_0^2}\left[\sin\varphi - \sin(\theta+\varphi)\right] \tag{4}$$

Rotation of the chair continues until $\omega = 0$ which gives maximum tilting $\theta_{\max}$ as

$$\theta_{\max} = \sin^{-1}\left(\frac{\omega_0^2 k_0^2}{2gc} + \sin\varphi\right) - \varphi \tag{5}$$

For complete toppling of the chair for this mode of motion, we need $\theta + \varphi = \frac{\pi}{2}$, which yields

$$\omega_{0,\min} = \frac{1}{k_0}\sqrt{2gc(1-\sin\varphi)} \tag{6}$$

Equation (6) determines the minimum initial angular velocity required to topple the chair for the mode considered. When $\omega_0 < \omega_{0,\min}$, toppling does not take place and the chair returns to the original state $\theta = 0$ (Figure 10).

Rotation about the Vertical z-axis. With reference to Figure 10, the equation of motion of the chair for chair rotation about the z-axis is given by

$$I_{zz}\ddot{\theta}_z = -\frac{\mu}{2}mg\sqrt{\ell_1^2+\ell_2^2} \tag{7}$$

Integrating (7) with the initial conditions $\theta_z(0) = 0$, $\dot{\theta}_z(0) = \dot{\theta}_{z0}$, we have the total rotation $\theta_{z,\max}$ about the z-axis as

$$\theta_{z,\max} = \frac{I_{zz}\omega_{z0}^2}{\mu mg\sqrt{\ell_1^2+\ell_2^2}} \tag{8}$$

Equation (8) determines the total angular rotation of the chair due to the impact.

Motion due to Blast Induced Flood Loading. In developing the equations that govern this mode of the chair motion, we assume that the blast loading is purely horizontal, and that the significant loading comes from the back rest area of the chair. This indicates that the arm, leg etc. do not contribute a significant part of the loading. Let us consider a blast wave of overpressure p_b comes from an angle θ with respect to the $x-$axis and measured positive for rotation along the positive $z-$axis (11). Then the direction of the blast load is $\vec{i}_b = \begin{pmatrix} \cos\theta_r \\ \sin\theta_r \\ 0 \end{pmatrix}$

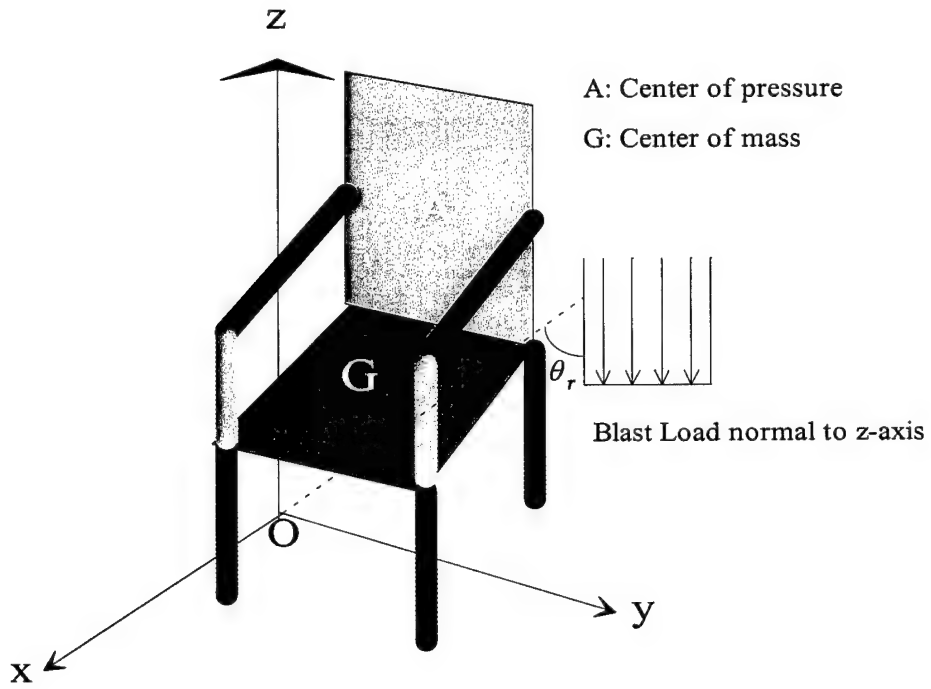


Figure 11. Blast load on Chair Model

The total impulsive load on the chair is $\vec{F}_b = p_b A_b \cos\theta_r \vec{i}_b$ and the total impulse is

$$\vec{I}_b = p_b A_b t_b \cos\theta_r \vec{i}_b \tag{7}$$

where t_b is the time duration of the blast load.

The initial chair velocity can be calculated from $\vec{v}_{b0} = \frac{\vec{I}_b}{m} = \begin{pmatrix} v_{b0x} \\ v_{b0y} \\ 0 \end{pmatrix}$. Solving the initial value problem using the same equations as (1a), we have the distances x_{bd}, y_{bd} the chair moves along the $x-$ and $y-$axis as

$$\begin{aligned} x_{bd} &= \frac{V_{b0x}^2 \,\mathrm{sgn}(I_{bx})}{2\mu_k g} \\ y_{bd} &= \frac{V_{b0y}^2 \,\mathrm{sgn}(I_{by})}{2\mu_k g} \end{aligned} \tag{8}$$

while there is no translation along the $z-$direction.

To calculate the rotation of the chair along the axes, we use the equations similar to the case of the point impact load discussed earlier [Equations (7) and (8)]. We replace the impulse moment as $\vec{M}_b = \overrightarrow{GA} \times \vec{I}_b$. The initial conditions are replaced by equations similar to (2) where the moment is replaced by the impulsive moment $\vec{M}_b$.

2.4.2 Desks and Tables

Similar to the chair model we also have developed a desk and table model that will used as one of the furniture in our visualization simulation. A schematic of the desk model is shown in Figure 12. The desk is assumed to be made of oak; and front and bottom openings, if any, have been ignored in calculating the dynamic behavior of the desk. A schematic of the STRICOM desk model is shown in Figure 12. For a specific case, the desk orientation and the blast load direction with respect to the axes system is given below.

Stricom Desk Model

Desk Sides: Side 1-4 starting from xz-plane to others by rotation about pozitive z-axis

Side 1 outward normal direction cosines $N1 := \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}$

Side 1 outward normal direction cosines $N2 := \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$

Side 1 outward normal direction cosines $N3 := \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

Side 1 outward normal direction cosines $N4 := \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}$

Blast Load Direction $N := \begin{pmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{2} \\ 0 \end{pmatrix}$

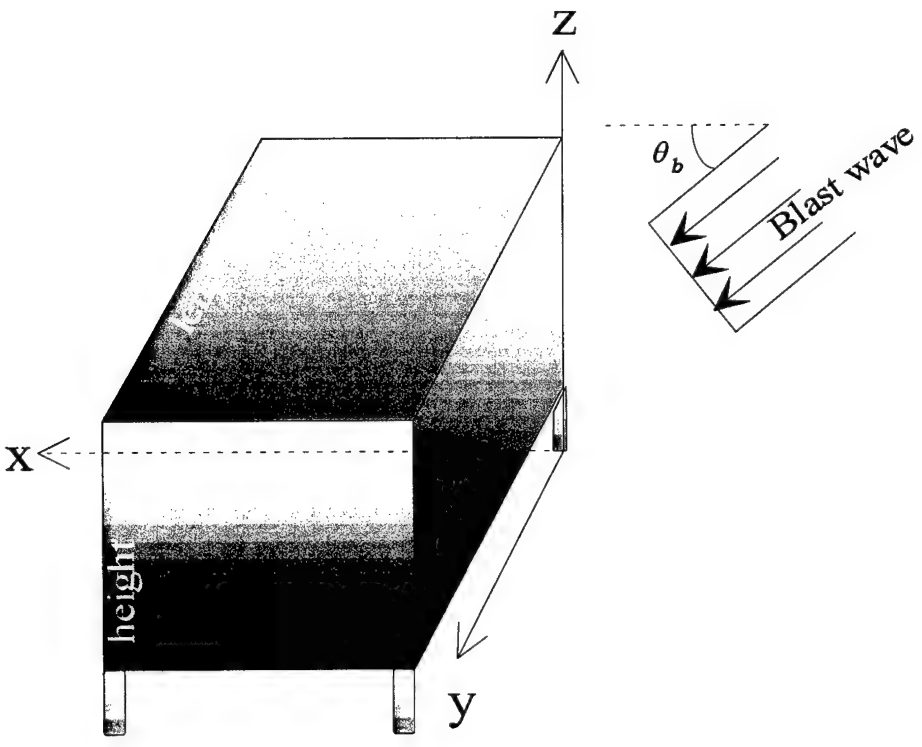


Figure 12. Schematic of Desk Model

Angles made with the sides

With side1 $\theta 1 := \text{acos}(N1 \cdot N)$ or $\theta 1 = 2.618$ $\theta 1d := \theta 1 \cdot \frac{180}{\pi}$ or $\theta 1d = 150$ degress

With side2 $\theta 2 := \text{acos}(N2 \cdot N)$ or $\theta 2 = 1.047$ $\theta 2d := \theta 2 \cdot \frac{180}{\pi}$ or $\theta 2d = 60$ degress

With side3 $\theta 3 := \text{acos}(N3 \cdot N)$ or $\theta 3 = 0.524$ $\theta 3d := \theta 3 \cdot \frac{180}{\pi}$ or $\theta 3d = 30$ degress

With side4 $\theta 4 := \text{acos}(N4 \cdot N)$ or $\theta 4 = 2.094$ $\theta 4d := \theta 4 \cdot \frac{180}{\pi}$ or $\theta 4d = 120$ degress

θ-Vector

$$\theta := \begin{pmatrix} \theta 1 \\ \theta 2 \\ \theta 3 \\ \theta 4 \end{pmatrix} \qquad \theta = \begin{pmatrix} 2.618 \\ 1.047 \\ 0.524 \\ 2.094 \end{pmatrix} \quad \text{in radians}$$

$$\theta d := \begin{pmatrix} \theta 1d \\ \theta 2d \\ \theta 3d \\ \theta 4d \end{pmatrix} \qquad \theta d = \begin{pmatrix} 150 \\ 60 \\ 30 \\ 120 \end{pmatrix} \quad \text{in degrees}$$

Calculate Projected Area (lengths are in inch) and Areas in sq.inch

height:= 20 length:= 60 depth := 40

area1 := depth·height area1 = 800

area2 := lengthheight $\text{area2} = 1.2 \times 10^3$

area3 := area1 area3 = 800

area4 := area2 $\text{area4} = 1.2 \times 10^3$

Area-Vector

$$A := \begin{pmatrix} \text{area1} \\ \text{area2} \\ \text{area3} \\ \text{area4} \end{pmatrix} \qquad A = \begin{pmatrix} 800 \\ 1.2 \times 10^3 \\ 800 \\ 1.2 \times 10^3 \end{pmatrix}$$

Projected Area proj_area = 1.293×10^3 **sq.in** along the blast directior

Overpressure pbl := 2.0 **psi**

Load Time tbl := 0.1 **seconds**

Impulse due to Blast Load Ibl := proj_area·pbl·tbl **or** Ibl = 258.564

Desk Mass mdesk := 50 **lbm**

Impulsive Velocity $vbl := \frac{Ibl}{mdesk}$ vbl = 5.171 **in/sec**

Coefficient of Friction $\mu := 0.2$

Linear Distance the Desk Moved $Ld := \frac{vbl^2}{2 \cdot \mu \cdot 32.2 \cdot 12}$ **or** Ld = 0.173

The differential equations of motion used in the above calculation are similar to the equations (1a) used in the STRICOM chair model.

SECTION 2 APPENDIX. CODE LISTING

Codes Written in C

```
STR_TEST CPP          783  12-03-98  4:20p str_test.cpp
AIRBUL~1 CPP        9,940  04-14-00  6:43p Airbull Gtype Version3.cpp
AIRBULL  CPP       29,222  01-19-99  5:50p AIRBULL.cpp
AIRBUL~2 CPP        8,107  01-05-99  4:05p Airbull_C.cpp
AIRBUL~3 CPP       15,479  01-12-99  5:30p Airbull_C_2.cpp
AIRBUL~4 CPP       18,107  01-15-99  4:22p Airbull_C_3.cpp
AIRBUL~5 CPP       27,516  01-20-99  4:03p Airbull_C_4.cpp
AIRBUL~6 CPP       37,672  03-02-99  3:18p Airbull_C_5.cpp
AIRBUL~7 CPP       38,443  03-09-99  6:52p Airbull_C_6.cpp
AIRBUL~8 CPP       42,786  03-29-99  6:15p Airbull_C_7.cpp
AIRBUL~9 CPP       42,907  03-30-99  4:37p Airbull_C_8.cpp
AIRBU~10 CPP       40,854  02-29-00  4:38p Airbull_C_Gerry.cpp
BULL_I~1 CPP        2,800  12-14-98  4:29p bull_info.cpp
BULL_I~2 CPP        8,107  12-18-98  5:09p bull_info_2.cpp
FRAME_~1 CPP        1,312  02-10-99  4:29p frame_correction.cpp
LIN_AN~1 CPP        1,584  03-01-00 12:02p Lin_Ana_Gm.cpp
LIN_AN~2 CPP        2,536  04-06-00  5:22p Lin_Ana_Gm_AC.cpp
PTR_TO~1 CPP          329  12-07-98 12:57p ptr_to_ptr.cpp
RKNLR2   CPP        5,846  01-06-99  4:19p Rknlr2.cpp
RKNLR2_2 CPP        7,144  01-07-99  6:16p Rknlr2_2.cpp
AIRBU~11 CPP        9,567  04-13-00  6:02p Airbull Gtype Version2.cpp
C_CODES  SCM            0  02-28-01 10:46a c_codes.scm
```

Codes Written in MATHCAD

```
NLSPR4   MCD        6,312  12-01-94  2:24p NLSPR4.MCD
GLASSF~1 MCD       29,106  01-31-01  5:50p Glass Fragment Analysis2.mcd
GLASSB~1 MCD       25,453  01-30-01  4:05p Glass Break Location3.mcd
GLASSB~2 MCD       32,190  01-30-01 12:56p Glass Break Location2.mcd
GLASSB~3 MCD        6,591  01-23-01  4:43p Glass Break Location.mcd
MASSVE~1 MCD        6,604  10-27-00  1:42a Mass Velocity Extension.mcd
GLASSF~2 MCD       25,957  08-29-00  3:22p Glass Fragment Analysis.mcd
GLASSF~3 MCD        6,152  08-24-00 11:47a Glass Fragments Code.mcd
RIGIDB~1 MCD       12,086  03-02-00 12:54p rigid body motion.mcd
AIRBUL~1 MCD       14,753  03-17-99  6:20p Airbull Nonlinear.mcd
AIRBUL~2 MCD       22,086  03-17-99  4:27p Airbull Check Code 3.mcd
AIRBUL~3 MCD       20,103  03-16-99  6:15p Airbull Check Code 2.mcd
WEAKNO~1 MCD       18,497  03-15-99  6:06p weak nonlinearity.mcd
AIRBUL~4 MCD       18,779  03-09-99  4:08p Airbull Check Code.mcd
AIRBUL~5 MCD       15,529  03-04-99  4:49p Airbull_Dir_cosines.mcd
PRESSURE MCD        6,525  01-17-95  3:06p PRESSURE.MCD
NLNRNEW2 MCD        6,877  01-03-95  7:46p NLNRNEW2.MCD
BULLET   MCD       11,482  12-22-94  7:08p BULLET.MCD
NLNRNEW  MCD        7,873  12-09-94  7:24p NLNRNEW.MCD
GLASSF~4 MCD       12,184  02-12-01 11:46a Glass Frag Mean Properties.mcd
MCAD     SCM            0  02-28-01 10:48a MATHCAD Code Listing
```

Codes Written in FORTRAN

```
BULPENT2 IN            832  04-25-97  4:09p BULPENT2.IN
AIRBULL2 FOR         9,471  08-28-96  2:06p AIRBULL2.FOR
AIRBULL3 FOR        18,447  09-04-96  1:15p AIRBULL3.FOR
AIRBULL4 FOR        20,952  09-05-96  1:21p airbull4.for
AIRBULL5 FOR        21,317  09-06-96  1:33p AIRBULL5.FOR
AIRBULL6 FOR        21,632  09-06-96  5:02p AIRBULL6.FOR
AIRBULL7 FOR        30,897  03-05-99  6:12p AIRBULL7.FOR
AIRBULL8 FOR        30,922  03-05-99  6:15p airbull8.for
AIRBULL9 FOR        31,461  03-09-99  3:44p AIRBULL9.FOR
BONEGEL  FOR        11,860  12-05-95  1:44p BONEGEL.FOR
BONEGEL0 FOR        11,076  11-20-95  3:19p BONEGEL0.FOR
BONEGEL2 FOR        12,481  01-23-96  4:26p BONEGEL2.FOR
BONEGELA FOR        18,533  05-22-96  5:15p BONEGELA.FOR
BONEGELN FOR        12,534  05-15-96  4:56p BONEGELN.FOR
BONEGELT FOR        20,614  05-28-96  2:02p BONEGELT.FOR
BONGEL   FOR        47,560  04-25-97  3:20p BONGEL.FOR
BONGELT2 FOR        21,433  06-19-96  3:16p bongelt2.for
BONGELT3 FOR        27,250  06-27-96 11:53p BONGELT3.FOR
BONGELT4 FOR        37,075  12-11-96  4:16p BONGELT4.FOR
BONGELT5 FOR        40,162  02-03-97  6:36p BONGELT5.FOR
BONGELT6 FOR        40,712  02-04-97  5:05p BONGELT6.FOR
BONGELT7 FOR        44,819  03-12-97 12:19p BONGELT7.FOR
BONGELT8 FOR        47,604  06-25-97 11:55a BONGELT8.FOR
BTBP     FOR         1,004  01-27-93  7:12p BTBP.FOR
BULLPEN  FOR        15,635  03-29-96  4:54p BULLPEN.FOR
BULLPEN2 FOR        15,967  04-15-96  6:00p BULLPEN2.FOR
BULLPENT FOR        19,728  07-19-96  4:46p BULLPENT.FOR
BULLPE~1 FOR        20,322  01-27-97  4:02p BULLPENT2.FOR
BULLRICO FOR        17,326  09-06-96  4:50p BULLRICO.FOR
BULPENT2 FOR        20,286  01-29-97  6:04p BULPENT2.FOR
BULRICO2 FOR        20,268  08-29-96  4:48p BULRICO2.FOR
FRAGMENT FOR         9,132  03-26-97 12:05p fragment.for
AIRBULL  IN            135  12-10-97  4:36p AIRBULL.IN
AIRBULL2 IN            211  09-17-96  1:39p AIRBULL2.IN
AIRBULL3 IN            585  09-04-96  5:29p AIRBULL3.IN
AIRBULL4 IN            514  09-04-96  6:23p airbull4.in
AIRBULL5 IN            514  09-06-96 12:47p AIRBULL5.IN
AIRBULL6 IN            513  09-11-96  5:29p AIRBULL6.IN
AIRBULL7 IN            445  03-02-99  2:44p AIRBULL7.IN
AIRBULLN IN            574  03-24-99  1:10p airbulln.in
BONEGEL  IN            450  02-05-96  5:03p BONEGEL.IN
BONEGELA IN            596  05-20-96  3:36p BONEGELA.IN
BONEGELO IN            451  12-05-95  1:38p BONEGELO.IN
BONEGELT IN            596  05-24-96  2:19p BONEGELT.IN
BONGELT2 IN            596  06-19-96  3:13p BONGELT2.IN
BONGELT3 IN            684  06-27-96 10:03p BONGELT3.IN
BONGELT4 IN            650  07-31-96 11:30a BONGELT4.IN
BONGELT5 IN            649  02-12-97  6:04p BONGELT5.IN
BONGELT8 IN            768  07-23-96  3:45p BONGELT8.IN
BTBP     IN             31  01-28-93 12:23p BTBP.IN
BULLET   IN            768  07-23-96  3:45p BULLET.IN
BULLIMP1 IN            792  04-25-97  4:09p BULLIMP1.IN
BULLIMP2 IN            832  04-25-97  4:09p BULLIMP2.IN
```

```
BULLIMP3  IN            832  04-25-97   4:09p BULLIMP3.IN
BULLPEN   IN            343  05-22-96   5:15p BULLPEN.IN
BULLPENT  IN            787  06-07-96   5:47p BULLPENT.IN
BULLRICO  IN            606  09-06-96   5:08p BULLRICO.IN
AIRBULL   FOR         8,059  12-10-97   4:38p AIRBULL.FOR
AMIYA     SCM             0  02-28-01  10:48a amiya.scm
CURVES    H           1,235  09-06-00   1:25a Curves.h
FORT      2          16,222  09-14-00  11:25a fort.2
FORT      5             728  08-30-00   9:31p fort.5
411       DAT           700  09-06-00  12:20p 411.dat
418       DAT           728  08-30-00   9:29p 418.dat
FORT5     DAT           728  09-06-00   7:58a fort5.dat
GLASS1    DAT             0  09-13-00   2:01p glass1.dat
GLASS2    DAT           727  09-14-00  11:25a glass2.dat
1                         0  09-14-00  11:25a 1
FORT5                     0  09-07-00   6:01p fort5
CURVEROO  F           7,499  09-06-00   3:51a Curveroo.f
CURVESET  F           4,572  09-06-00  10:58a Curveset.f
DETONATE  F          10,874  09-10-00   2:20a Detonate.f
FLOOKUP   F           4,527  09-06-00   8:44a Flookup.f
FORCE     FOR         1,935  09-12-00   3:43p force.for
GQ1DRFPA  F           2,856  09-06-00   3:13a Gq1drfpa.f
GQ2DRF    F           3,968  09-06-00   3:11a Gq2drf.f
GQITER    F           4,632  09-06-00   3:08a Gqiter.f
GQLEGPOL  F             940  09-06-00   3:04a Gqlegpol.f
GQREGFAL  F           6,916  09-06-00   3:03a Gqregfal.f
GQSAMPLE  F           3,198  09-06-00   2:55a Gqsample.f
INTABL    F          17,728  09-06-00   3:49a Intabl.f
IRFUN     F           2,289  09-06-00   6:27a Irfun.f
ISHELL    F             104  09-06-00   2:14a Ishell.f
LINTERPO  F           2,571  09-06-00   2:14a Linterpo.f
NVCURVS   F           1,477  09-06-00   2:09a Nvcurvs.f
PRFUN     F           2,885  09-06-00   8:53a Prfun.f
PSHAPE    F             545  09-10-00   3:06a pshape.f
SESAME    F           2,537  09-06-00   3:42a Sesame.f
SWITCHLL  F           1,706  09-06-00   3:55a Switchll.f
SYSBEEP   F             205  09-06-00   1:56a Sysbeep.f
USERINPU  F           5,432  09-06-00  11:27a Userinpu.f
VENTEDBL  F             777  09-07-00   5:27p Ventedbl.f
ARRAYLIM  H             158  08-04-00  10:23a ARRAYLIM.H
FORT      1          17,458  09-14-00  11:25a fort.1
INPAR     H           2,044  08-24-00   8:48p Inpar.h
INTRO     H             482  09-06-00   1:31a Intro.h
STANDOFF  H             169  09-06-00   1:31a Standoff.h
FORT5     XLS        58,880  09-07-00   6:24p fort5.xls
VENTEDBL  OPT        43,520  09-14-00  11:57a Ventedbl.opt
VENTEDBL  PLG         2,781  09-12-00   3:45p Ventedbl.plg
B         TAB             299  08-24-00  12:32p B.TAB
I0        TAB             352  07-07-00   2:16p I0.TAB
I06       TAB             335  07-07-00   2:15p I06.TAB
IR        TAB             346  06-30-00   9:56a IR.TAB
IS        TAB             333  06-29-00   2:32p IS.TAB
IX        TAB             552  08-09-00   3:36p IX.tab
PQS       TAB             412  07-05-00   2:04p PQS.TAB
PR        TAB             328  06-30-00   9:56a PR.TAB
PS        TAB             348  06-27-00   2:22p PS.TAB
PX        TAB             448  08-09-00   3:36p PX.tab
```

```
REFLRAT  TAB            203  08-09-00   3:43p ReflRat.tab
T0       TAB            438  07-06-00   3:49p T0.TAB
T06      TAB            404  07-07-00   2:01p T06.TAB
TA       TAB            379  08-24-00  12:51p TA.TAB
TD       TAB            367  08-24-00  12:51p TD.TAB
VENTEDBL DSW            539  09-07-00   6:24p Ventedbl.dsw
VENTEDBL DSP          6,827  09-07-00   6:24p Ventedbl.dsp
FRAGMENT PLG          1,994  01-31-01  12:36p FRAGMENT.PLG
FRAGME~1 PLG            816  02-12-01  10:51a Fragment2.plg
FRAGMENT IN             204  02-12-01  12:43p Fragment.in
FRAGMENT FOR          2,242  01-31-01  12:32p Fragment.for
FRAGMENT DSP          3,253  01-31-01  12:08p Fragment.dsp
FRAGME~1 FOR          3,522  02-12-01  10:52a Fragment2.for
FRAGMENT OPT         43,520  01-31-01  12:37p Fragment.opt
FRAGMENT DSW            539  01-31-01  12:38p Fragment.dsw
FRAGME~1 DSP          3,265  02-12-01  10:32a Fragment2.dsp
FRAGME~1 OPT         43,520  02-12-01  11:46a Fragment2.opt
FRAGME~2 FOR          4,582  02-12-01  12:08p Fragment3.for
FRAGME~1 DSW            541  02-12-01  11:46a Fragment2.dsw
FRAGME~2 DSP          3,265  02-12-01  11:47a Fragment3.dsp
FRAGME~2 PLG            816  02-12-01   4:44p Fragment3.plg
FRAGME~3 FOR          4,872  02-12-01   4:47p Fragment4.for
FRAGME~1 IN             281  02-12-01   4:42p Fragment4.in
FRAGME~2 OPT         43,520  02-12-01   4:44p Fragment3.opt
FRAGME~2 DSW            541  02-12-01   4:44p Fragment3.dsw
FRAGME~3 DSP          3,265  02-12-01   4:45p Fragment4.dsp
FRAGME~3 PLG            816  02-12-01   4:47p Fragment4.plg
FRAGME~1 OUT            721  02-12-01   4:47p FRAGMENT4.OUT
FRAGME~3 OPT         43,520  02-12-01   4:50p Fragment4.opt
FRAGME~3 DSW            541  02-12-01   4:50p Fragment4.dsw
DEBUG          <DIR>        02-26-01   5:00p Debug
CHAMBER  SCM              0  02-28-01  11:21a chamber.scm


417      DAT            156  06-30-00   2:41p 417.DAT
417      OUT          8,713  08-29-00   4:36p 417.OUT
417FREE  DAT            117  06-29-00   3:07p 417FREE.DAT
417FREE  OUT          9,101  06-30-00  10:32a 417FREE.OUT
417REFL  DAT            129  06-30-00   9:42a 417REFL.DAT
417REFL  OUT          9,204  06-30-00  10:32a 417REFL.OUT
43CRATER DAT            333  07-07-00   9:18a 43CRATER.DAT
43CRATER OUT          8,947  07-07-00   9:22a 43CRATER.OUT
43FREEAI DAT            311  07-07-00   9:16a 43FREEAI.DAT
43FREEAI OUT          8,947  07-07-00   9:26a 43FREEAI.OUT
44       DAT            313  06-30-00   2:59p 44.DAT
44       OUT          9,448  06-30-00   2:59p 44.OUT
45       DAT            332  07-05-00  10:07a 45.DAT
45       OUT          9,448  07-05-00  10:14a 45.OUT
46       DAT            277  07-05-00  10:49a 46.DAT
46       OUT          8,846  07-05-00  10:49a 46.OUT
B        TAB            308  06-29-00   2:23p B.TAB
CURVES   H              551  07-27-00  10:03a CURVES.H
DETONATE FOR          8,947  08-29-00   4:59p Detonate.for
FLOOKUP  FOR          5,320  07-27-00  10:02a FLOOKUP.FOR
GRENADE  MAK          2,006  06-29-00   3:17p GRENADE.MAK
GRENADET MAK          1,609  06-29-00   3:21p GRENADET.MAK
INPAR    H              776  07-27-00  10:13a INPAR.H
```

```
INTABL    FOR        21,478  07-27-00 10:02a INTABL.FOR
INTRO     H             639  07-27-00 10:03a INTRO.H
IR        TAB           346  06-30-00  9:56a IR.TAB
IS        TAB           333  06-29-00  2:32p IS.TAB
ISHELL    FOR           341  07-27-00 10:02a ISHELL.FOR
PR        TAB           328  06-30-00  9:56a PR.TAB
PS        TAB           348  06-27-00  2:22p PS.TAB
Q         TAB           330  06-30-00 10:30a Q.TAB
SESAME    FOR         3,337  07-27-00 10:03a SESAME.FOR
SWITCHLL  FOR           818  07-27-00 10:03a SWITCHLL.FOR
SYSBEEP   FOR           516  07-27-00 10:03a SYSBEEP.FOR
TA        TAB           376  06-29-00  2:34p TA.TAB
TD        TAB           357  06-27-00  2:24p TD.TAB
TEMPFILE                586  07-27-00 10:03a TEMPFILE
UBAR      TAB           313  06-30-00 10:31a UBAR.TAB
UBARS     TAB           291  06-21-00  2:11p UBARS.TAB
GRENEDE   FPW            89  08-23-00  2:38p GRENEDE.FPW
GRENEDE   FMK         1,749  08-04-00  1:53p GRENEDE.FMK
GRENEDE   WSP           363  08-23-00  2:38p GRENEDE.WSP
GRENADE   OBJ         2,382  08-04-00  1:54p GRENADE.OBJ
GRENADE   DSP         4,927  08-29-00  6:28p Grenade.dsp
GRENADE   PLG         1,708  08-30-00  3:50p Grenade.plg
GRENADE   OPT        43,520  08-30-00  3:50p Grenade.opt
USERIN~1  FOR         3,992  07-27-00 10:15a Userinput.for
ARRAYL~1  H             130  07-27-00 10:03a Arraylimits.h
CURVES~1  FOR         4,312  07-27-00 10:19a Curvesetup.for
GRENADE   FOR         1,050  08-29-00  4:35p Grenade.for
417A      OUT         8,713  08-29-00  5:00p 417a.out
417B      OUT         8,713  08-29-00  5:37p 417b.out
GRENADE   DSW           537  08-29-00  6:28p Grenade.dsw
GLASSF~1  FOR         1,171  11-08-00  4:39a GLASSFRAG.FOR
GRENADE   SCM             0  02-28-01 11:21a grenade.scm
```

SECTION 3.0

CHARACTER PROCESS SIMULATOR

The runtime version of the first-generation STRICOM character simulator (*MRCMAN)* has been structured to interface to the main simulation through a socket and stream protocol. This allows the real time portion of the simulation to proceed while *MRCMAN* does casualty analysis. This also provides a mechanism for handling multiple simultaneous hits on the same person, while making multitasking and/or multithreading implementations transparent to the real time system. The other advantage of this implementation is that it is more portable. *MRCMAN* can be run in the same system as the real time environment, or it can be hosted on any available machine on the local network.

MRCMAN is an ongoing effort to simulate the effect on a person of penetrating wounds. The current version has been ported to the Windows NT environment as a stand-alone application. The package allows the user to define the body dimensions and postures of the simulated person and define the projectile geometry, construction, and striking conditions (velocity, obliquity, and yaw). After the initial conditions are set the trajectory through the body is evaluated.

This evaluation takes two forms. In the simplest form the wound assessment is bimodal and describes whether the soldier is an *operational casualty*. At this level of assessment, the details of the wound are not fully considered. Rather, at this level of assessment the issue is the body region affected and the depth of projectile penetration (i.e., is it sufficient to enter a body cavity for the anatomical region affected?) For the extremities, the issue is the probability of striking a bone or severing a major blood vessel or nerve plexus as a function of penetration depth and projectile residual velocity. From this perspective a leg wound, which would prevent an infantryman from walking has the same result as a wound that would have killed him, there is one less infantryman available to complete a mission.

On a more detailed level of assessment, MRCMAN can also assist in conducting a medical casualty assessment. The code describes the trajectory of the impacting round through the body recording each tissue disrupted by the projectile. In this higher resolution mode a more detailed assessment can be made of the wound, the level of medical support required to treat the wound, and with the appropriate medical support even a prognosis. Casualty reduction potential or other mission metric can be evaluated by comparing multiple mission simulations using a soldier equipped with different equipment enabling different capabilities. For example, the same mission can be simulated with and without body armor or body armor with different coverage areas to determine the effectiveness of body armor on mission completion.

In the STRICOM/MRC simulation an embedded interface was provided for MRCMAN. The GUI version requires that the user specify the body position, round type and trajectory. The embedded version does not have the GUI, but rather receives data from other applications that define the body size, posture, and position, as well as projectile data. In the context of the

simulation, when an Event occurs, and it is determined that the round hits a person in the simulation, the event processor passes a data structure to MRCMAN identifying the event, defining the body posture and the projectile data. MRCMAN then evaluates the casualty status of the person hit. For the purpose of the simulation, a person who has become a casualty is put into a "fall down" behavioral mode. A non-casualty would continue as before. The more detailed data on tissue damage is stored to disk and identified by the event.

As a matter of possible future development, real life does not have this binary state behavior. Rather, a soldier who is a casualty may still be able to move to cover and provide fire support to the rest of the team. And a soldier who has been injured, but is not a casualty, is less effective than someone who has not been injured, effecting overall team performance. This level of modeling could also be extended to model other factors, such as fatigue and thermal stress, and speed of movement.

A top-level flow chart of the MRCMAN first generation character simulator is shown below in Figure 13.

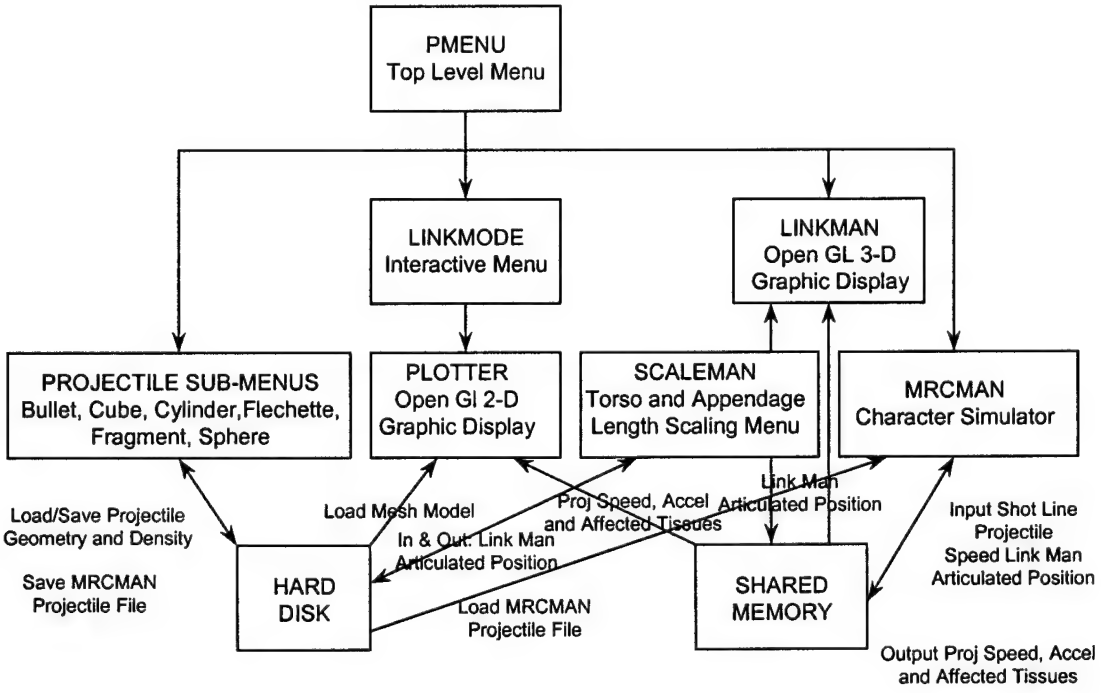


Figure 13. First-Generation Character Simulator (MRCMAN)

SECTION 4.0

BALLISTIC TESTING

The objectives of the ballistic testing conducted in the second phase of this program were threefold. The first objective was to provide a database for validation of the projectile terminal ballistics model development. The second objective was to provide empirical data so that coefficients in terminal ballistics algorithms representing the interaction between various projectiles and materials could be determined. The third objective was to provide reference images for texture mapping of target damage from ballistic impact events. In this section, the ballistic test data is provided in total for both phases of the program. Section 4.1 begins with a review of the Phase I ballistic data followed in Section 4.2 by a review of the Phase II ballistic testing approach and data.

4.1 PHASE I BALLISTIC DATA

Phase I ballistic testing employed weapons and ammunition shown in Table 2 on typical interior wall, ceiling, floor, door, and glazing materials. Typical Phase I ballistic targets are shown in Figure 14. All ballistic testing during Phase I was conducted at a range of eight-feet from the target with normal striking obliquities. The exit velocity of the striking projectile through the rear surface of the target was determined, debris fields were video taped, ejecta was captured in witness plates, and the target entrance and exit holes were photographed. These tests were conducted to capture phenomenology, yield preliminary data on projectile velocity loss after target penetration (see Table 3), and provide texture maps.

Table 2. Ammunition used during Phase I STRICOM Ballistic Tests against Targets Representative of Residential Construction

Gun	Round	Total Mass (grams)	Diameter (inches)	Projectiles in Salvo	Muzzle Velocity (fps)
12 ga Shotgun	Rifled Slug	28.4	0.73	1	1600
12 ga Shotgun	#4 Cu-plated Buckshot	50.5	0.24	41	1210
12 ga Shotgun	#00 Lead Buckshot	29.6	0.33	9	1325
9mm Handgun	FMJ Parabellum	7.5	0.36	1	1155
7.62mm Rifle	FMJ	8.1	0.31	1	2100

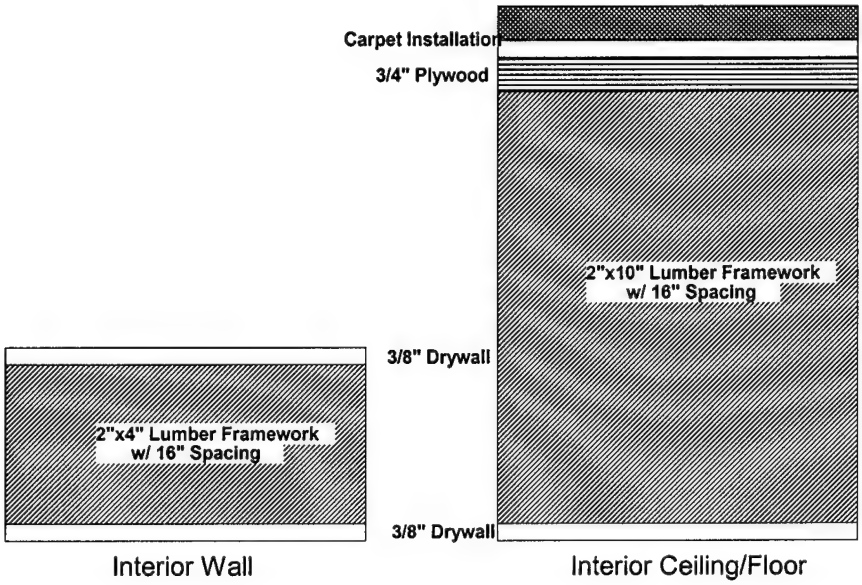


Figure 14. Phase I Ballistic Target Cross Sections

Table 3. Results from Phase I Ballistic Experiments

SHOT #	TARGET TYPE	ROUND	MUZZLE VELOCITY	RESIDUAL VELOCITY	VELOCITY LOSS
01	INT. WALL	SLUG	1600	1500	6%
02	INT. WALL	#00-B	1325	1150	13%
03	INT. WALL	#4-B	1210	1090	10%
04	INT. WALL	9mm	1155	1100	5%
05	INT. WALL	7.62mm	2100	2080	0%
06	INT. DOOR	#4-B	1210	1050	13%
07	INT. DOOR	#00-B	1325	1080	18%
08	INT. DOOR	SLUG	1600	1470	8%
09	FLOOR	#00-B	1325	980	26%
10	FLOOR	SLUG	1600	1480	8%
11	FLOOR	#4-B	1210	950	21%
12	FLOOR	9mm	1155	1050	9%
13	CEILING	9mm	1155	1090	6%
14	CEILING	#4-B	1210	700	42%
15	CEILING	#00-B	1325	650	51%
16	CEILING	9mm	1155	1070	7%
17	CEILING	SLUG	1600	1470	8%
18	1/8" STD GLASS	#00-B	1325	N/A	N/A
19	1/4" PLATE GLASS	#00-B	1325	N/A	N/A

4.2 PHASE II BALLISTIC TESTING APPROACH AND DATA

A major objective of the Phase II ballistic testing was to acquire ricochet and penetration data. The test matrices associated with these two sets of data are shown in Table 4. The empirical data required to calibrate the penetration algorithms in the AIRBULL software are shown in Table 5.

Table 4. Initial Phase II Test Matrices

Ricochet					
Target	.30-06	9mm Para.	12ga Slug	#4 Shot	00 Buckshot
Dry Wall					
Block	X	X			
Cement	X	X			
Plywood					
Hollow Door					
Solid Door					
Steel Door	X	X			
Glass	X				

Penetration				
.30-06	9mm Para.	12ga Slug	#4 Shot	00 Buckshot
X	X	X	X	X
X				
X	X	X	X	X
X	X	X	X	X
X	X	X	X	X
X	X	X	X	X
X	X	X	X	X

Table 5. Empirical Data Required for Penetration Algorithms

Debris		7.62	5.56	9-mm	12 g slugger	#4 Shot	00 Shot
X	**Dry Wall**	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin
	Block*	Vc2 > Vmax	Vc2 > Vmax	Vc2 > Vmax	Vc2 > Vmax	Vc2 > Vmax	Vc2 > Vmax
	Cement	Vc2 > Vmax	Vc2 > Vmax	Vc2 > Vmax	Vc2 > Vmax	Vc2 > Vmax	Vc2 > Vmax
X	**Plywood**	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin
	Hollow Wood Door	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin
	Solid Wood Door						
X	**Steel Door**						
X	**Glass**	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin	Vc1 = Vmin

* rebar and mortar filled

Vmax	Muzzle velocity of weapon
Vmin	Minimum velocity of interest
Vc1	Velocity at which projectile just penetrates target
Vc2	Velocity at which projectile just perforates target

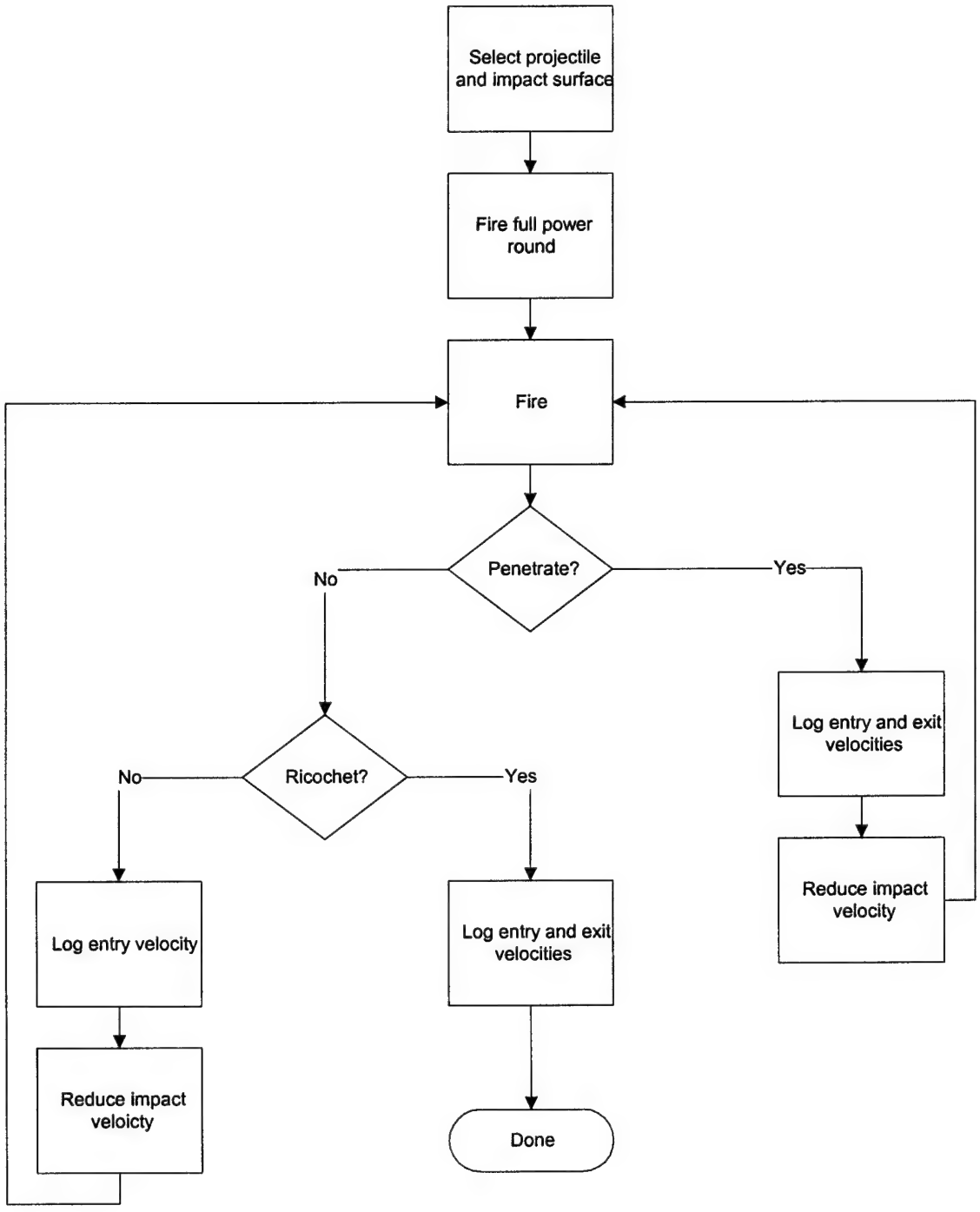


Figure 15. Initial Phase II Testing Methodology to Develop Table 5 Data

The weapons delineated in the program scope (see Table 1) can be resolved into three classes. The M16A1 (5.56 mm M193 ball), M16A2 (5.56 mm M855 ball), M249 (5.56 mm M855 ball), and the M60 (7.62 mm M80 ball) are Full Metal Jacket (FMJ) sharp ogival rounds. These rounds were simulated using the FMJ .30-60. The M9 (9mm M822 ball) and 12 gauge slugger rounds are parebellums and the 00 shot and 04 shot fired from the 12 gauge shotgun are deformable lead spheres.

As discussed above, the ballistic experiments can be divided in to two test categories – ricochet and penetration tests as shown in Table 4. In the ricochet tests the striking and ricochet angles were measured and the incident and rebound bullet velocities determined (see Table 6). The "x" in the matrix cells determines the relevant projectile-target combinations that were tested.

The penetration experiments were designed to determine three things. First, where relevant, the threshold striking velocity for which complete perforation of the target occurs needed to be determined. Second the exit velocity was measured for various striking velocities and finally, for relevant projectile target combinations, the debris/ejecta field was mapped. In terms of the Table 5 nomenclature it was the intent of the ballistic testing to: (1) Validate the use of the V_{c1} and V_{c2} terms to approximate impact behavior, (2) Empirically determine values for V_{c1} and V_{c2} for various projectile target combinations, and (3) Validate the hypothesis that penetration is a cosine-squared function of the angle of incidence.

In the ricochet testing different rounds at different muzzle velocities were fired at targets through a range of incident angles. During the test firing additional photographs and measurement were made to record the extent of damage to the target material resulting from the impact. This data was used to model the real time visualization of the impact event.

The original ballistic testing methodology designed to acquire the relevant data is shown in Figure 15. While the methodology outlined in Figure 15 would have provided the needed data, the labor required to step through each iteration of the loop was excessive. To accelerate testing progress, multiple targets were used in sequence to modify the iterative nature of the testing.

The original test plan required firing one round through a single layer target, then changing the powder charge and firing again. Instead, the testing methodology was changed so that one round was fired at several targets spaced along the trajectory (see Figure 16). The residual velocity of the round was measured between target "layers". One example of a target configuration that was used was firing at a sample of drywall construction, with five samples of the wall spaced out every five feet. The number of layers and the range of muzzle velocities however was a function of the resistance of the target material to penetration (e.g., fewer tests need to be run on steel doors than hollow core wood doors, the steel door stops the round much sooner).

The ballistic experiments involving stacked targets included shooting layered targets with three walls (double stack of drywall) followed by a steel door and solid wood door. The residual velocity between each target pair was determined and the target damage was digitally photographed. The debris between the plywood and drywall was mapped using high-speed digital video. The projectiles that were tested included 5.56mm, #4 shot and #00 shot at a normal striking angle. Selected images from this testing is shown in the Section 4.0 Appendix.

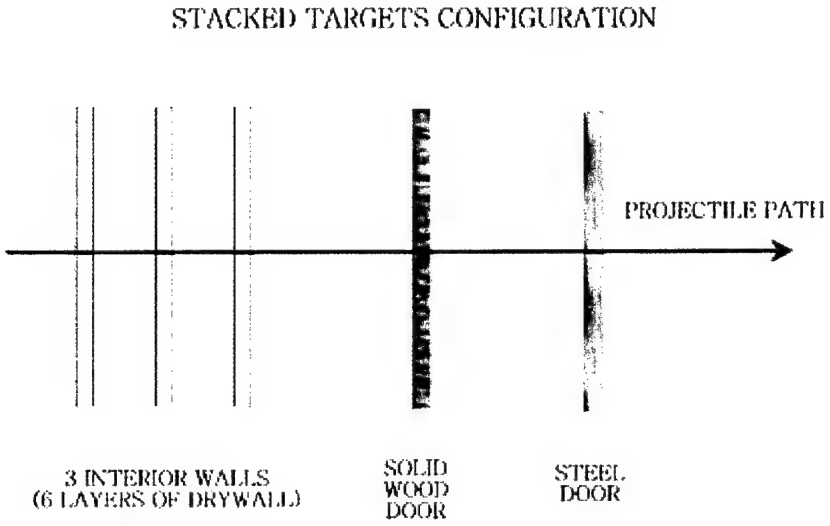


Figure 16. Phase II "Stacked Target" Configuration

5.56mm V_s=3200fps

target	V_r (fps)
hollow door	~3200
solid door	3010
steel door	1780
3 int. walls	2340

stacked targets

round	$V_{s\ (fps)}$	V_r (fps) 3 int. walls	solid door	steel door
5.56mm	3200	2340	1450	940
9mm	1155	810	700	defeated
12ga slug	1600	1280	930	540
12ga 00-B	1325	940	350	defeated
12ga #4-B	1210	590	defeated	n/a

Figure 17. Typical Results from Stacked Target Configuration

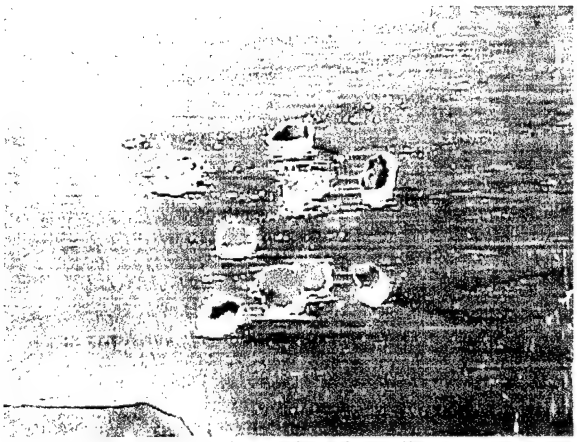
Table 6. Phase II Ricochet Test Results

Run #	Round	Mass (gm)	Muzzle Velocity (fps)	Target	Angle of of Incidence (q)	Angle of Rebound (f)	Residual Velocity (fps)	Comment
980311-01	22LR	2.6	1150	steel	18	0		
980311-02	22LR	2.6	1150	steel	23	2		"round may have hit ""dimple"" in plate"
980311-03	22LR	2.6	1150	steel	23	1		
980311-04	22LR	2.6	1150	steel	23	1		
980311-05	22LR	2.6	1150	steel	18	1		
980311-06	22LR	2.6	1150	cement	18	1		
980317-01	9mm	7.5	1155	steel	18	1		slight crater in target
980317-02	9mm	7.5	1155	steel	22	1		small frag. @ 1.375 (2.3 deg.)
991118-01	12ga Slug	28.3	1600	steel door	90		1360	
991118-02	12ga #00	29.6	1325	steel door	90		780	
991118-03	12ga #4B	50.6	1210	steel door	90			defeated round
991118-04	9mm	7.5	1155	steel door	90		790	
991118-05	9mm	7.5	1155	steel door	15	<2	1050	
991120-06	12ga #4B	50.6	1210	solid wood door	90		500	
991120-07	12ga #00	29.6	1325	solid wood door	90		980	
991120-08	12ga Slug	28.3	1600	solid wood door	90		1490	
991120-09	9mm	7.5	1155	solid wood door	90		1040	
991203-10	9mm	7.5	1155	solid wood door	17	<2		
991203-11	12ga Slug	28.3	1600	solid wood door	13			too near edge
991203-12	12ga #4B	50.6	1210	solid wood door	13	<2		
991203-13	12ga Slug	28.3	1600	solid wood door	13	6		
991208-14	12ga #00	29.6	1325	steel door	21	<2		
991208-15	12ga Slug	28.3	1600	steel door	21	<2	1170	
991208-16	12ga #4B	50.6	1210	steel door	21	<2	1090	
991208-17	9mm	7.5	1155	steel door	15	7	1010	
991208-18	12ga Slug	28.3	1600	glass	13			penetration
991208-19	12ga #4B	50.6	1210	glass	13			penetration
991208-20	12ga #00	29.6	1325	glass	13			penetration
991208-21	9mm	7.5	1155	glass	13			penetration

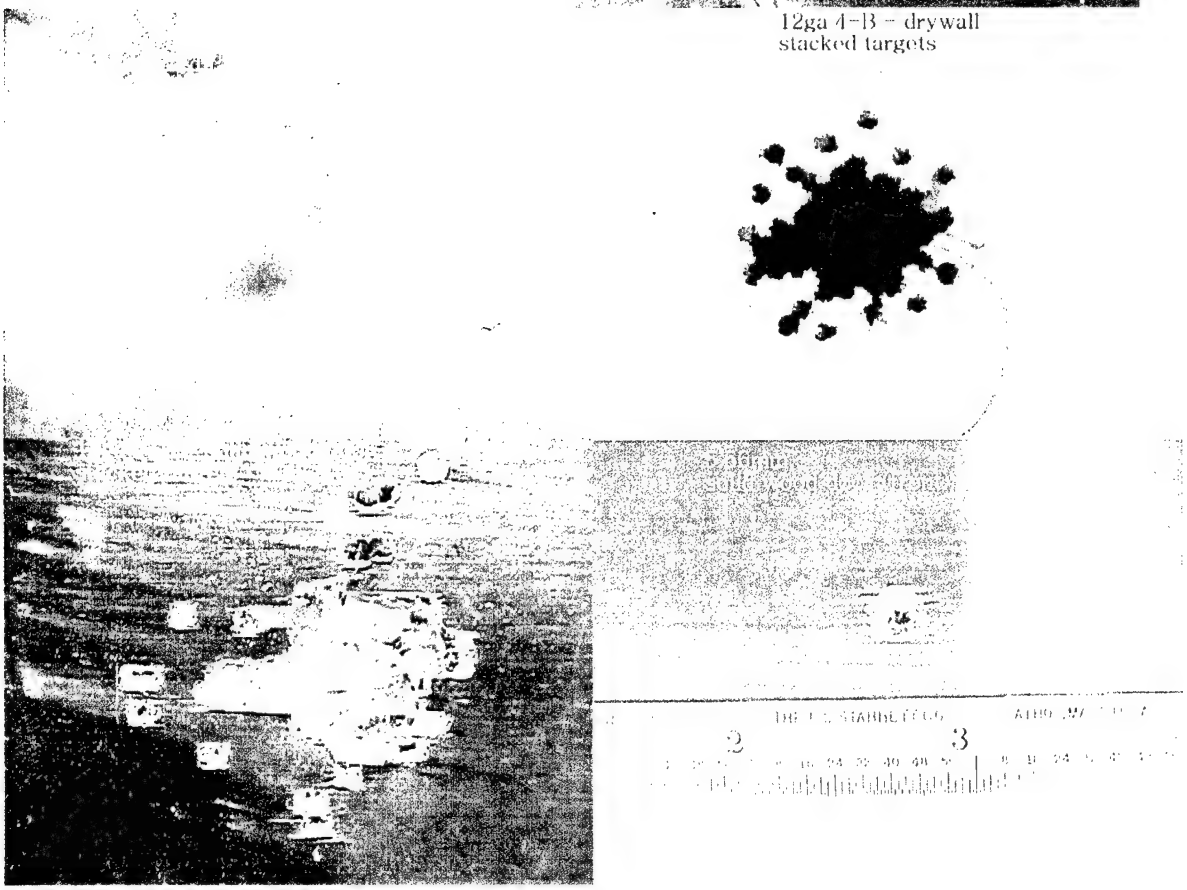
SECTION 4.0 APPENDIX

SELECTED IMAGES FROM PHASE II BALLISTIC TESTING

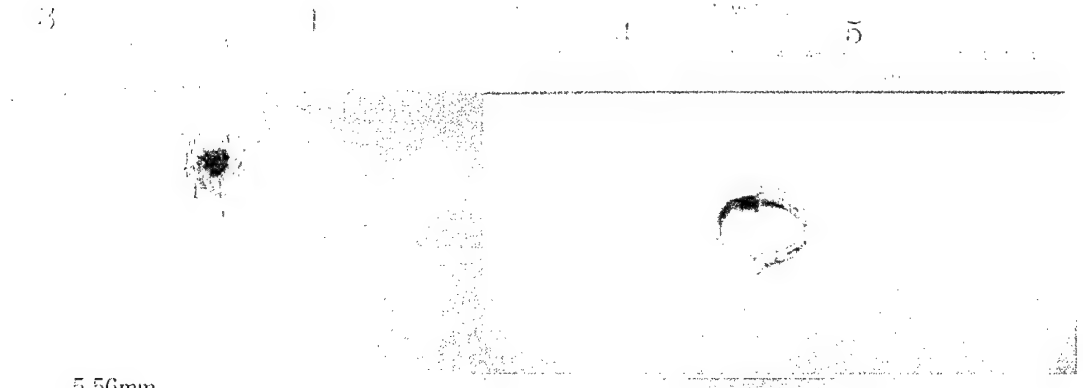
12ga 00-B – drywall stacked targets



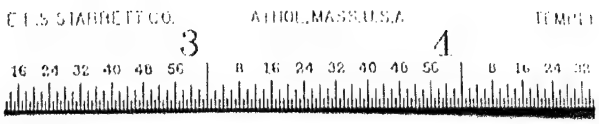
12ga 4-B – drywall stacked targets



5.56mm
steel door (front)

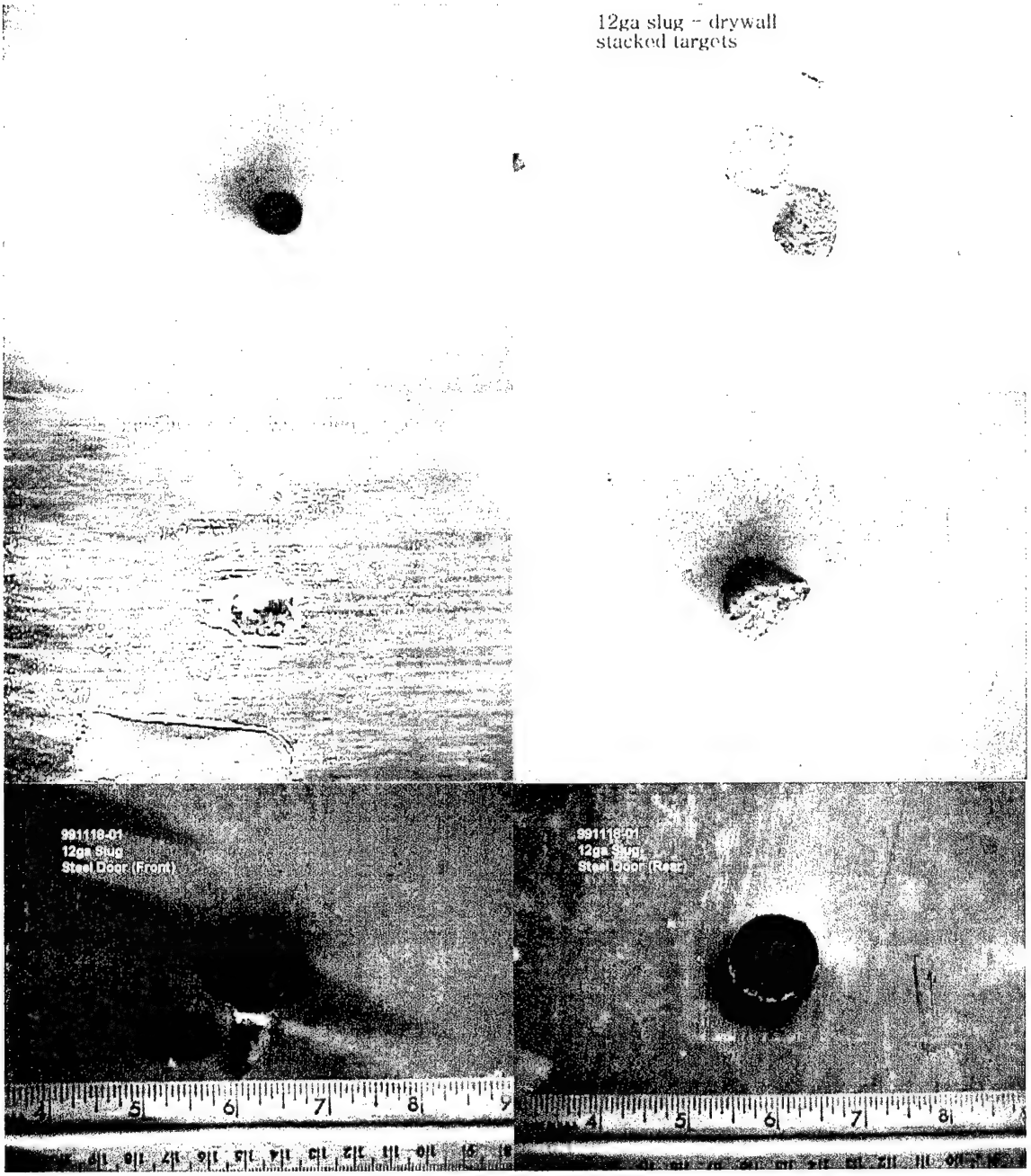


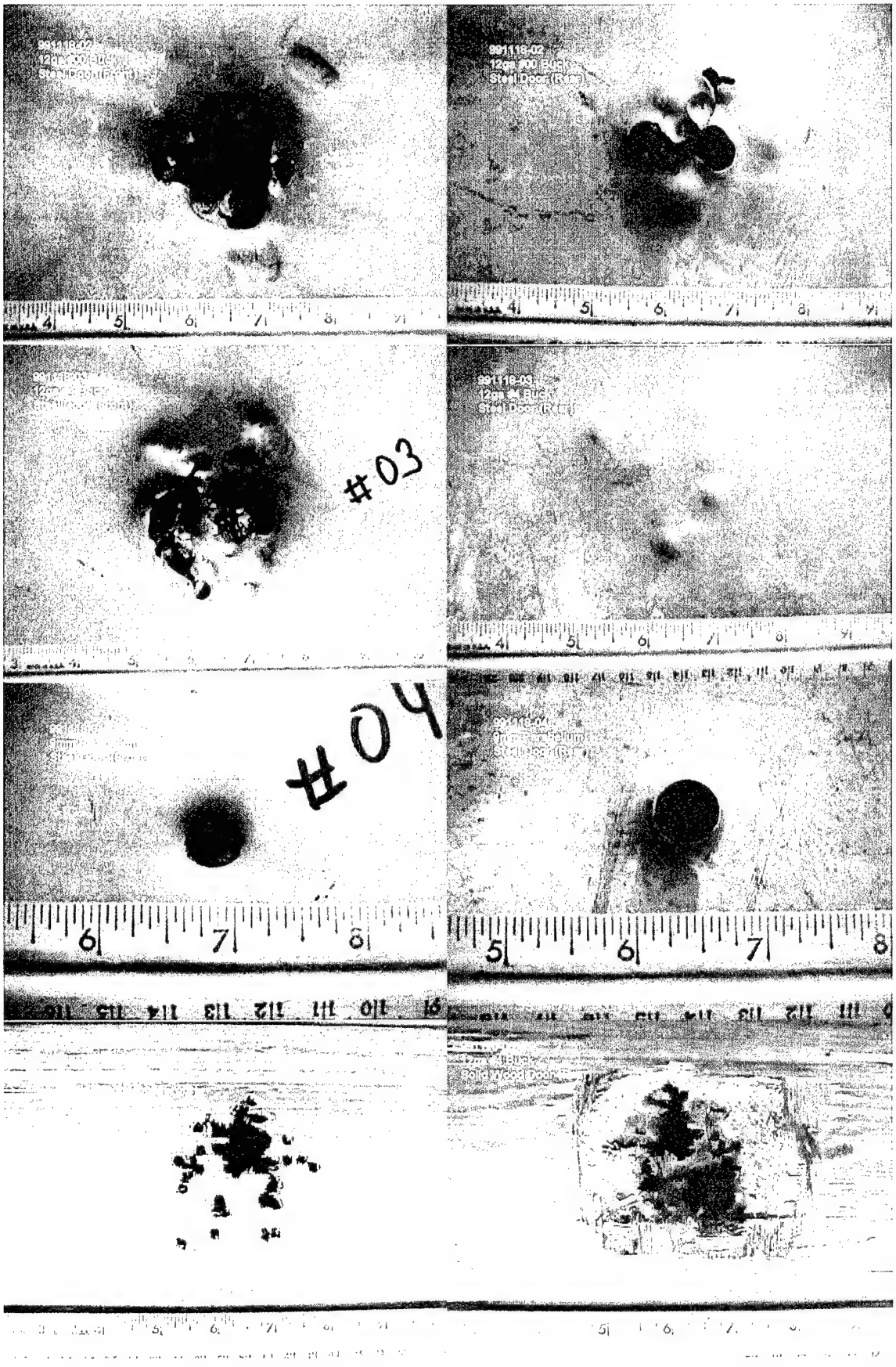
5.56mm
steel door (rear)

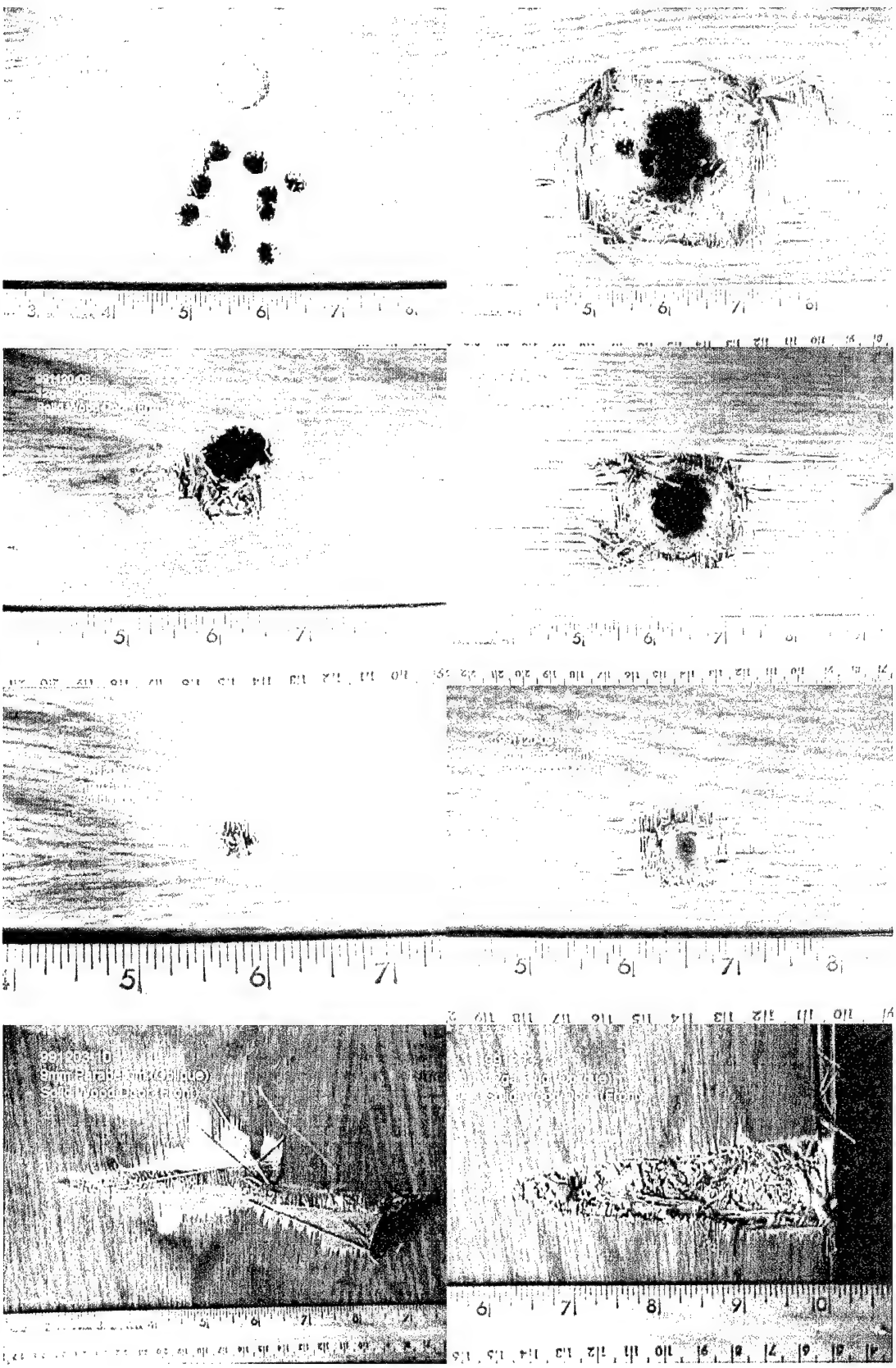


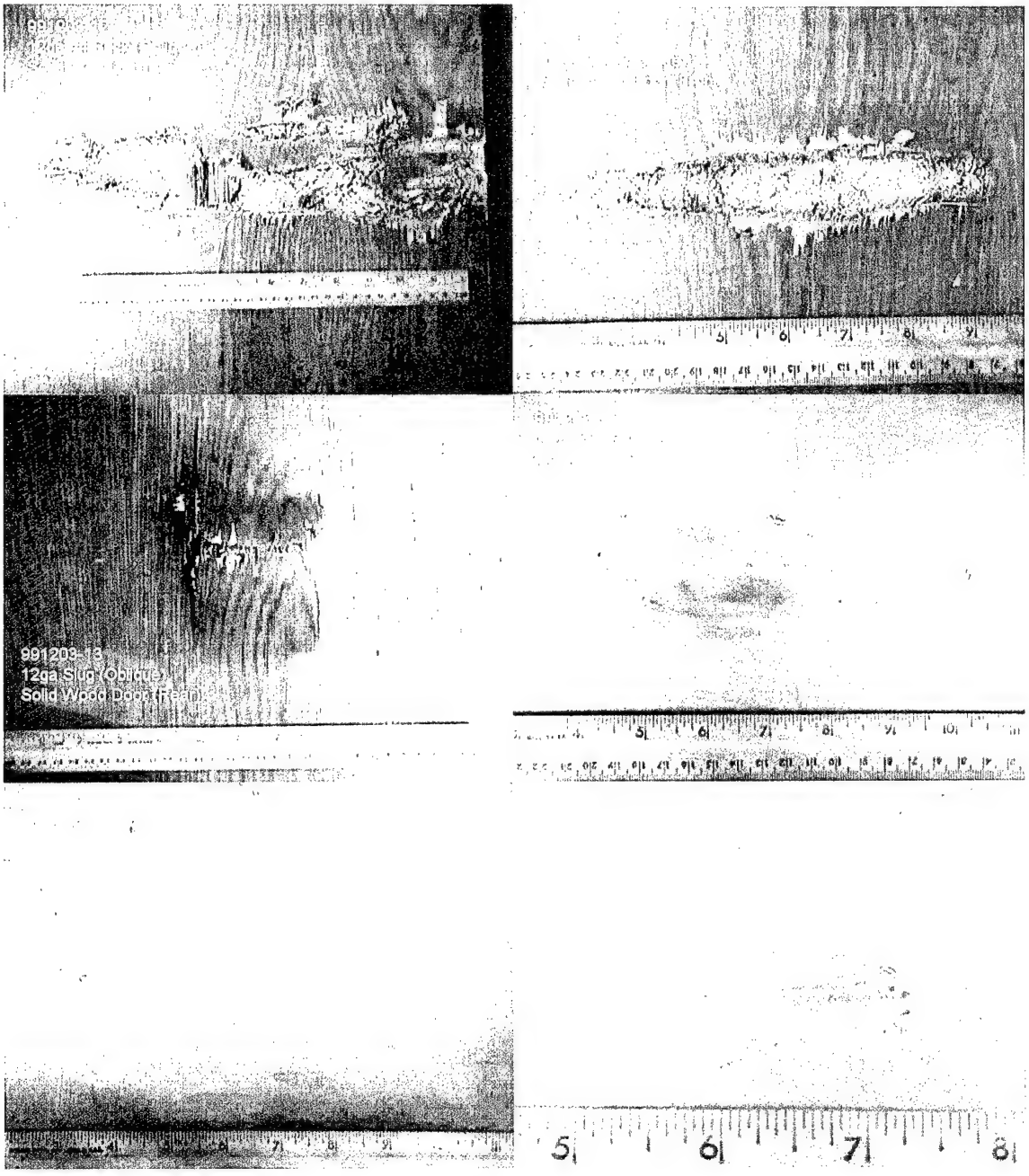
9mm - drywall
stacked targets

12ga slug – drywall
stacked targets




#03
#04




991203-13
12ga Slug (Oblique)
Solid Wood Door (Rear)

SECTION 5.0

SOFTWARE ARCHITECTURE AND VISUALIZATION SYSTEM

The goal this effort is to provide a working software foundation to allow training in a realistic combat environment and analysis of the actions that take place in that environment without the expense and hazards associated with a live fire exercise. One of the key components missing in previous simulators is the ability to simulate the damage to the immediate environment from weapons effects. A traditional Virtual Reality (VR) simulation would show characters moving through a "pristine" environmental database with visual effects representing weapon effects superimposed on the environment. In traditional VR simulations the occurrence of weapon effects does not actually modify the three-dimensional database from which the simulated environment is generated. As a result, the interaction of characters with traditional VR environments must be scripted. These scripts can have a sophisticated decision tree associated with them resulting in a compelling simulation. However, the range of outcomes is established *a priori* and the interaction is therefore constrained to the pre-existing scripts.

Our approach has been to create a generalized simulation that includes the dynamic environmental changes produced by weapon effects and that can exist in a distributed computer environment. Sub-real-time algorithms that described the physics of the respective interaction govern the environmental changes to the database. Outcomes are not scripted or known *a priori* so there is a complete unconstrained free-play of characters with the environment. As a result unexpected results and mistakes can and do occur. In addition, due to the unconstrained nature of character interaction, the simulation can also be used as a virtual prototyping tool where notional equipment can be tested.

A generalized approach has been dictated by the very nature of the problem, that almost anything can be considered a weapon and can be used to damage almost any other object. So other than the first few seconds (prior to the changes in the database produced by weapon effects); the internal logic of a traditional VR simulation and the current STRICOM simulation are completely different. Given that most of the effort has focused on the internal logic of the simulation, where possible we have tried to exploit the capabilities of Commercial Off-The-Shelf (COTS) visualization tools where appropriate.

There are three facets to the current software architecture and visualization system discussed below in Sections 5.1 through 5.3 – visual modeling, structural simulation, and visual simulation. Section 5.4 discusses software integration including the COTS software utilized. Section 5.5 discusses the detailed software requirements of the modules developed for the STRICOM simulation.

5.1 VISUAL MODEL

The modeling effort has presented a series of interesting problems. Development of the weapon effects simulation for the second floor for the McKenna Town Hall highlighted the fact that although our general approach was well founded, the labor resources necessary to generalize the weapon effects models so that they could be directly imported to other simulations exceeded program resources. This was known during Phase I and was restated in the Phase II proposal. At the center of this problem was the need to represent and codify structural data in what is primarily a visual database. It was assumed from the beginning, that the structural database and the visual database would have to be created and maintained as a single logical construct. If an independent structural database (e.g., CAD/CAM derived) and visual database (photo textured) were used, there would be a high probability of the two having conflicting requirements, particularly during the dynamic updating process, assuming that the two databases were ever compatible to begin with.

Another issue was maintaining the "appropriate" level of complexity in the database for this type of simulation. In a combat simulator, the exact placement of 2x4 studding is rarely an issue. But whether a wall is built of wood frame and drywall versus reinforced concrete will have a major impact on tactics driven by the level of available hard cover. With this in mind, most CAD/CAM oriented databases contain too much data, while most visual databases contain too little. The normal rules for designing a visual database are to provide a level of visual detail that is appropriate for the nominal view distance and visual acuity provided by the system. Within this context, the simpler the model, the better the system performance is. Hence the partitioning of most visual databases has very little relationship to the physical reality of the modeled object.

One of the more significant "lessons learned" has been that significant enhancements to existing modeling tools will have to be developed before any large scale simulation environments could be rendered using the approach in this project. Consider a couple of simple examples. Take a room with wood frame and dry wall construction. If there are any windows or doors, there are major structural discontinuities around the windows, doors and corners. And there is also the issue of load bearing versus non-load bearing walls. Another case would be a concrete wall. Depending on construction technique, the walls could be very homogeneous or have regularly space structural members, while most of the wall is relatively "soft." While all of the walls may appear to be the same within the visual simulation, the differences in response to weapons could be significant. To provide the needed level of fidelity, each major homogeneous structural element must have a corresponding set of geometric constructs in the visual database. Currently, this can be done by "slicing and dicing" an appropriate visual database – but this process is extremely labor intensive.

As an adjunct to the modeling process, spatial relationships have to be identified and encoded. To accomplish this, two fundamental conceptual objects were used: Objects and Spaces. An *Object* is any logically homogeneous entity or a non-homogeneous collection of other Objects. Hence, a building is an Object made up of progressively higher detailed Objects (e.g., a door). A *Space* is any reasonably defined volume in the simulation environment. So, once again, a building would be in Space, with each room within the building being a sub-space. For any given space, it will have two lists of Objects associated with it, boundary objects and internal objects. A boundary Object is one that defines the Space itself (e.g., a wall). One of the characteristics of

boundary Objects is that they usually are part of more than one Space. Boundary Objects are also nominally fixed. Internal Objects are everything else. An internal Object is nominally in only one space at a time, though it may exist in several (e.g., a man standing in a doorway).

The main intent of using the Space and Object constructs was to reduce the runtime complexity of the simulation. When a round if fired in a room, the room defines a space. That space contains a known list of objects. Only those objects will be considered for a possible impact until the round leaves its current space (i.e., the bounding volume of the room).

5.2 STRUCTURAL SIMULATION

In order to maintain some level of consistency in the coding process, C++ classes and inheritance were used to define the physical objects in the simulated environment. The base class "Object" has various sub-classes that allow for modeling complex behaviors. For example, there are classes – *Weapon* and *Ordnance*. For the purposes of this simulation, a Weapon is an object that carries and fires Ordnance, which is, in turn, capable of doing damage to the simulated environment. So, there is a sub-class of Weapon to model an M-16, and a sub-class of Ordnance to model a 5.56-mm round. Most explosives would be sub-classes of both Weapon and Ordnance.

Any Ordnance is ballistic, explosive, or both. So, once a Weapon is triggered, and the current ordnance (e.g., the round in the chamber) is/maybe fired; the code is implemented to allow various failure modes (e.g., misfire, failure to feed) and a potential for catastrophic failure (e.g., a destructively overloaded round or a squib followed by a live round). Once ordnance has been fired, it either follows a ballistic trajectory until it impacts something, or it explodes and the shock wave moves outward effecting all objects that it passed through until such time as the intensity of the blast wave attenuates to a minimal level.

It is the function of the *Event process*, which runs remotely from the visual process, to execute the time dependent analysis of the effect of all active Ordnance in the simulated environment. This time dependency allows for more accurate representations of the physical behavior of the various objects. As a simple example consider a window. Normal window glass is not bullet or explosion proof, but it will deflect the trajectory of a round fired through it. If two riflemen fire at the same window, the rounds will go through the window leaving a hole. But most of the window will still be there and continue to deflect any further rounds. However if a grenade is launched/thrown at the window and a rifle round is fired at the window, the time dependent nature of the problem could be critical in determining casualties behind the window. If the shock from the grenade reaches the window first, the bullet may not have any glass to pass through, and hence not be deflected. If the bullet arrives first, the window will be pre-cracked and the blast wave may create significantly more dangerous debris.

5.3 VISUAL SIMULATION

The current version of the visual simulation is based on the Performer interface developed by Silicon Graphics (SGI) and is being run on an Indigo 2 High Impact system. While this is a

reasonable development environment, it is not representative of *state of the art* rendering systems. Given the current state of visual systems, a suggested follow-on activity would be to port the software to a Linux system with Performer support.

In the mean time, several simplifications have been made to allow faster updates. The most significant constraint is initially limiting the visualization to the immediate space(s). If the eye point is in a room, only the room will be rendered. If the eye point is in a volume where two or more Spaces overlap, the objects in all of these spaces will be passed to the graphics engine. Given a more powerful visualization system, other logic can be enabled that will allow the user to see through several spaces as would be expected in the actual environment.

One of the functions of the Event process is to maintain a list of all the currently valid graphics data. In an HLA context, the Event process "owns" all of the graphics data. When an event occurs, and the geometry changes (e.g., holes created, objects deformed/destroyed) the Event process updates the geometry and passes the revised data to all of the visualization stations.

One of the basic concepts in the visualization of events is that most events have time constants much faster than the refresh rate of the visualization system. When an explosion goes off, anything in the immediate environment is hit in less than 10 milliseconds, which would correspond to a 100 Hz update rate on the visualization system. Or, given the same 10 milliseconds, a 5.56-mm round will cover approximately 10 meters. Given that few rooms are more than 10 meters on a side, the muzzle flash and the impact effect should happen in the same frame.

A further simplifying assumption is that explosions at close range are disorienting. If an explosion is detonated (e.g., a "flash-bang") in close proximity to a character, the people who knew it was coming will be disoriented for a fraction of a second; the intended target will be disoriented significantly longer. The point being, that while an explosion can cause significant disruption to the environment, the bigger the explosion, the longer that the computer has to come up with a valid representation. Currently, a flash or whiteout image is provided.

5.4 SOFTWARE INTEGRATION

Four separate software components were integrated to form the system nexus. These components were the real time visualization system, the event processor, the MRCMAN character simulator, and the Motivate™software package. A multi process, multi platform approach is critical to any successful simulation environment for any time in the foreseeable future. The drive for greater realism will absorb all computer resources available at any given cost point. As a result, it is not economically feasible, and in many cases technically impossible, to provide a monolithic solution for a multi-player environment. Given that there was no over riding constraint imposed to create a single system design, each of these software components has a different history and addressed different operational requirements. The selection of each operating system environment was determined by a combination of factors, including cost, availability, and experience.

At this time, the visualization system is tied to SGI's Performer™ as a run time environment and Centric Software's Designer's Work Bench™ (DWB) as a visual modeling environment. At the outset of this project, the combination of an SGI workstation and Performer provided a solid, high performance graphics environment. With the porting of Performer from the Iris operating system to the Linux operating system, the visualization system can be ported to a non-SGI specific platform, though such a move is not currently planned or budgeted. The event processor is currently tied to Performer, and hence an SGI workstation environment.

5.4.1 Event Processor

From a High Level Architecture (HLA) perspective, the event processor "owns" every physical object in the simulation environment. While the event processor can give temporary ownership of an object to another process, it must always be able to pull back that control, e.g., a physical object, such as a door. In the nomenclature of Motivate, all moveable objects are "actors" e.g., the representation of a person is an actor as is the door to the room that the "person" is in. Under the control of an "actor" in Motivate, the door can be opened or closed. However, the processing of an event (i.e., discharging a weapon) may determine that the door was damaged or destroyed. In this case, the event processor must pull back control of the door, determine it's current state, and distribute that state information to the visualization system(s). It is this interaction with the visualization system that ties the event processor to Performer. The "master copy" of all positional and geometric data in the model database must be present in the event processor for it to evaluate the interaction of the objects in the immediate vicinity of an event. The most effective way to accomplish this was to use the Performer Loader. While there is nothing that constrains future implementation to using the model loaders from Performer, using the same loaders does provide a much higher level of confidence that both the visualization and event processes will be using the same initial model data.

The event processor serves two functions. The first function is an ongoing task as a clearinghouse for the simulation database, making sure that all visualization systems have a valid copy of the current "master copy." All actions in the simulated environment are routed through the event processor before the data is passed to the visualization systems. This mode of operation allows for assured transfer of data between all processes that are active in the simulation. The other task is the actual event processor that operates on non-linear occurrences. The combination of the database management functions with event processor allows the event processor to take ownership of any object at any time. When an event occurs, the event processor sends a message to the transient owner of an object directing it to release control. However, the event processor can immediately take control of the affected object and dump any incoming control data from the transient owner.

5.4.2 MRCMAN Character Simulator Integration

MRCMAN is an ongoing effort by Mission Research Corporation to model the effect of battlefield trauma to soldiers. Given a set of initial conditions defining body proportions, posture, protective equipment, type of projectile, striking conditions and location on the human body; the software package determines the type and amount of tissue damage that resulted. The current version of MRCMAN is hosted in a Windows environment.

5.4.3 Motivate™

Motivate is a run time and software development environment created by The Motion Factory. Motivate provides a rich environment for modeling physical behaviors (i.e., it can show the chicken crossing the road, but it can't tell you why the chicken did it). Motivate provides a foundation for goal based direction of either avatars or synthetic characters in the simulation environment. It permits current activities to be raised to a higher level of abstraction. Take a "simple" activity like "go through the doorway," Motivate will accept an initial mode (e.g., walking, crouched, or crawling), and a destination (i.e., the doorway). It then computes a path, taking into account any obstructions, and provides the frame to frame animation of the subject to represent the activity. All of the body part locations and joint rotations can be read off at each frame to provide the global data for both the visualization and MRCMAN. The Motivate development environment is Window NT and the run time environment is either Windows NT or Windows 98.

It should be pointed out that the flow of data to Motivate is bi-directional. An actor could be controlled by any one of four possible means. Vectors and state changes provide the simplest control data mechanism (e.g., a command might consist of a state change to standing, followed by a walk state and a command to walk to a given position). The event processor can determine that some event has altered the actor's position, joint rotations, or even physical attachments (e.g., an explosion may have shattered an object). These are the only mechanisms that will be available in the current project. However, future enhancements could include synthetic actors who are self-motivated (i.e., the actor "decides" to stand up, rather than being commanded to by operator intervention), and distributed control through an HLA interface (though this does raise an number of issues with respect to latency).

5.4.4 Network Communication

The integration process uses streams with TCP/IP to provide the needed interprocess communications. Streams were selected for a number of reasons. The most critical is that streams are applicable across all major platforms. The software technology represented by streams is independent of the transport layer that is used. It can work just as well in a multi-processor system as it does in a networked environment. And there is no assumption of any specialized hardware facilities present in the systems that are part of the simulation. At the same time it is reliable. A data gram protocol would be faster, but many of the critical messages are one-time transmissions, so reliable delivery of the data is critical for consistent results in the simulation environment. The final issue that lends itself to using streams is data consistency. Consider a squad level training simulation. Each person involved in the exercise must be presented with ground truth in real time. If this is not the case, inconsistencies in the training environment become a source of negative training.

A communications layer has been added to the original software architecture along with protocols for data validation of the data content and types, and conversion of data from the sending frame of reference to the receiving frame of reference. MRCMAN has been successfully modified and controlled by a remote test driver. The Motivate code is currently being modified to transmit current position and rotation data, and to receive input data to cause transitions in the state machine that controls actors (i.e., any object) in the Motivate environment.

When this integration is successful, the current test driver will be replaced with the combination of the event processor and real time visualization system. The most significant recent integration issue has been the implementation of a cross platform stream interface under the Windows operating system. The stream interface that is being used is a slight variation of a library of C functions that has been in use for over a decade. These functions serve as a simple wrapper for a fast, but low-level interface to the network under Irix/Unix.

5.5 SOFTWARE REQUIREMENTS

Requirements related to the notion of C++ classes (Section 5.5.1), depth complexity (Section 5.5.2), event processor efficiency (Section 5.5.3), mobility (Section 5.5.4), C++ object class (Section 5.5.5), C++ space class (Section 5.5.6) and Oct-tree structure (Section 5.5.7) are discussed below

5.5.1 C++ and C++ classes

The choice of C++ as the programming language resulted in a more concise, structured and transportable software package. However, this structure came at the cost of additional analysis and design effort. Two base C++ classes were employed in the STRICOM simulation to ease problems related to scene complexity. These classes were *space* and *object*.

5.5.2 Depth Complexity

On an SGI platform each polygon that faces the eye point and is within the viewing frustum is rendered. The determination of which polygon is rendered by a specific pixel is handled by a depth masking function. Hence, if the polygons are rendered in a back to front order, each pixel can be rendered multiple times. This is referred to as depth complexity. When the average depth complexity reaches a certain level, the hardware requires more time to render the image than is available in the target frame period. When this occurs, the system is said to be "fill bound" and the update frame rate is reduced. In the simple town hall simulation this is not a serious problem but is a factor in certain rooms with viewing angles that include large portions of the building. As additional buildings are added to the environment, management of depth complexity will become essential.

5.5.3 Event Processor Efficiency

The event processor efficiency must be maintained. When an event occurs, the volume affected by the event progresses either along a trajectory or radially outward. To efficiently determine interactions, the structural objects are spatially sorted. Otherwise, when an event occurs, the entire database must be checked for interactions.

5.5.4 Mobility

As a surface object (e.g., a person or a ground vehicle as opposed to a helicopter) moves through the database, it has to be supported. The supporting structure has to be determined for reasons of attitude, position and mobility. This would resolve issues such as the mobility of an APC moving over a solid road versus rubble versus mud. For dismounted infantry there is also the issue of

shifting surface (e.g., debris and rubble). Infantry also has to contend with moving through a damaged building with openings in the floor or possible structural collapse. Damaged buildings have a potential to collapse over a period having a very long time constant when compared to normally considered weapon events. This has to take into account both slow failures and fire effects. These issues are not currently within the scope of Phase II effort however.

5.5.5 C++ Space Class

A "Space" is a hierarchical construct defined by a bounding volume. A single root Space defines a simulation environment, which can then be subdivided into regions, and sub-regions, such as buildings. A Space may contain a list of Objects. These Objects are either internal to the Space, or are part of the defining boundary between two or more Spaces. A desk in a room would be an internal Object, while a wall could be a boundary object.

The purpose of the *space* class is to provide structure. The base class would have no intrinsic shape, sub-classes would be defined to provide effective implementation of geometry (e.g., rectangular prisms versus spherical). A "space" may or may not directly contain any "physical" objects. A space can have other spaces as "children". For example, one *master space* may include an entire MOUT site. This would provide a quick check for any events that occurred outside of the MOUT site, such as artillery.

5.5.6 C++ Object Class

An Object is a hierarchical construct that defines either a physical entity or conceptual representation. For example, Weapon is a class derived from Object. All distinct weapons (e.g., M16) have Weapon as a parent class. An example of a conceptual class is a Portal, which defines the connection between two Spaces when there is no physical object at the boundary. Portals facilitate the computation of line-of-sight and sound transmission.

An *object* is any physical entity in the simulation. This can be a person, vehicle, building or debris. Objects have implied or explicit logical structure. An object, such as a building that can be entered, would have a logical structure of rooms and passageways. These rooms would have a structure of atomic physical structural elements (e.g., wall elements, doors and windows).

5.5.7 Oct-Tree Structure

Within each "space" there would be a provision for an *oct-tree* structure. This provides a mechanism to localize effects and link sub-spaces. For example, given a master "space" for the MOUT, the oct-tree would initially provide differentiation of the northeast, northwest, southeast and southwest quadrants of the site. Within each of these quadrants, each block may be given it's own space. The individual buildings within the block would be sub-spaces of the block, with rooms being sub-spaces of the building. The streets would also be provided with a space, which in turn could be further subdivided.

Spaces in the same level of the hierarchy would tend to be adjacent, but more often will overlap and share objects. For example, given two adjacent rooms with a common wall, each room is a space. Each space extends to the far face of the common wall (i.e., you are not into the next room until you are all the way through the wall, regardless of which room you start in). The objective

is to have a controlled hierarchy that provides a quick mechanism for evaluating the local environment with more or less detail as needed.

To appreciate the implications of this system consider a "man" in a room. From the viewpoint of this person, he needs to be able to see the room and its contents. If the room is not a sealed, opaque structure, objects outside the current space may also be visible. If he starts an event, that event must interact with the objects in this space. If there is an opening, such as a door or window, or if the structures defining the space fail, the event can pass into another space. If the man moves across the floor, does the floor shift, or is there a void that he can fall through? What information does the simulation need to have about this space in order to provide this functionality?

*5.5.7.1 **Mobility***. The first piece of information is mobility. Any object that moves will have some form of ground loading, either in the form of a foot, wheel, track, or rotor wash. The first issue is a function of the simulation logic. The ground loading can only be applied to objects that are contained in the current space (though, rotor wash can have a secondary influence on the adjacent spaces). Something moving over soft ground leaves tracks, which, in turn, affects mobility. Walking on rubble is unstable, hence, it is low mobility and noisy. Moving over a damaged surface can generate additional noise and the possibility of further collapse. The second consideration is graphical. For a reasonable level of fidelity, the moving object must move in a rational manner relative to the surface which they are shown moving over (e.g., someone going up a flight of stair should step on the treads of the stairs, not look like they are gliding up a ramp). By knowing which "space" the "man" is currently in, the determination can be made as to which object(s) are supporting the man. The load bearing objects can then be evaluated for any changes in state. The position and stance of the object of interest can then be adjusted accordingly.

*5.5.7.2 **Visibility***. The next piece of information concerns visibility. Consider a "simple" case of an "L" shaped room on the upper floor of a building. This room has a door, which leads into a hallway, and a window that looks out over other buildings. The room would most likely be defined as a single parent space that serves as a bounding volume for the entire room. This parent could have three child spaces, one for each leg of the "L", and one for the intersection of the two legs. These three spaces might be further subdivided based on the amount of inter-visibility among the three spaces.

The sole purpose of adding sub-spaces is to reduce the number of objects that need to be rendered. During the analysis of the room, if it is determined to be simple enough, there may only be the parent space. However, if the room contains a large number of detailed objects, or it has a number of extended viewing areas, it may be necessary to define a tight grid of spaces to control the list of objects that are sent to the graphics hardware.

The list of viewable objects associated with the space is tied to a number of logical conditions. If the door is closed, the hallway is not visible, and therefore, is not rendered. If the drapes/blinds of the window are closed, or there is something else obscuring the window, the adjacent buildings are not visible. However, if the blinds are open, or gone, the other buildings are visible. The elevation of the current space relative to the surrounding buildings determines the list of other building/spaces that the viewer can see.

A very careful balance was required between automatic level of detail (LOD) switching and selective control of the visual content. Using conventional LOD switching, someone looking out a window at another building would see the outer surface of the building at some level of detail. However, for the class of applications that we are looking at, this is not adequate. Given a normal viewing angle and a range of 200 meters, a low level of detail representation for a building with windows is completely valid. However, replace the causal viewer with the view through the scope of a sniper's rifle, and then not only the high level of detail must be present, but the internal details of the room behind the window. This becomes a switch based on viewing frustum, range and screen resolution. How many pixels make up the window that we are looking through? When the number reaches a certain threshold (a function of the window size, range, field of view, viewing angle, and the resolution of the display window), and if the blinds are open, then the objects contained in the "target" room's space need to be enabled. Of the factors listed above to determine the switch point, only field of view and resolution are variable for one viewing space looking into another, and they do not change frequently. Therefore, the determination to provide greater detail can be made by a simple logical comparison.

The list of logical conditions is not fixed, but is dependent on past events. A wall that has been breached can be seen through. Immediately after an event has created a new line of sight option (e.g., a wall being breached), the visible object list for the space that the wall connected with would be added to the list for the current space. However, given this new, extended list of objects, it may be possible to see only a small number of the new objects from within the current space. Therefore, a background process was provided which refines the visibility list.

5.5.7.3 Events. The third group of operations on a space is related to events. When an event occurs, the event processor determines the trajectory of the event object (e.g., bullet, fragments, or blast wave), and computes the first objects that intersect the trajectory. Since the event occurred in a space, the trajectory must intersect with either an object that is contained in the current space, or it has to exit the space. If it exits the space, it enters another space, and the analysis continues. As objects are affected by the event, their properties, including visual models, are updated.

5.5.7.4 Objects. This brings us back to the topic of the *object* class. An object is any physical object, which may, or may not, have sub-objects. The base class does not constrain itself to rigid body transformations. An object has a position, attitude and the data that defines it. The object class provides the virtual functions that define the general operations that are applicable to any object in space. This includes the functions for setting position, rotations, updating visual state, and interactions with basic events. By defining these functions, a basic level of functionality is assured for any type specific sub-class as it is defined. These sub-classes have as many extending functions, which are class specific, as are required. As an example, assume that there is a *sheet_rock_wall* class and a *john_o* (a.k.a. MRCMAN) class. Both of these classes can have their 6-DOF values set. They can both be shot or be affected by an explosion. But the *john_o* class will eventually permit detailed examination and evaluation of the medical damage resulting for this event, while the *sheet_rock_wall* class will only update its structural integrity and visual state. With the addition of the Soldier System Command/MRC Phase II SBIR Second Generation Character Simulator effort, notional equipment can be added as sub-classes that have the appropriate functionality to represent the properties of the equipment.

Other base functions for the object class will consist of automatic dispatch and receipt of DIS (or other format) messages. Future expansion of sub-classes is largely unconstrained. As the

technology for autonomous agents improves, progressively more sophisticated versions of MRCMAN can be implemented while maintaining the same operational interface.

5.6 SOFTWARE ARCHITECTURE

The objective of this project is to demonstrate the effects of weapons in an urban combat environment, and to do so in real time. In an effort to make this project more widely useable, there has been a subtle but definite shift in the underlying software architecture from what was originally proposed. This has been prompted by the rapid evolution of the computer industry, which has opened a number of options that were not available at the time of the proposal. This section will cover the reasons for this shift (Section 5.6.1), implication of High Level Architecture (HLA) requirements on the software (section 5.6.2), reasons for revising the software architecture (Section 5.6.3), and the current software configuration (Section 5.6.4).

The problem that this project solves is to describe the dynamic nature of structures in a virtual urban combat environment. A traditional virtual environment would show characters moving through a "pristine" set. While this may be valid for the first few seconds of a simulation, the effect of ordnance can quickly and radically modify this.

This modification takes both an active and a passive form. In an urban environment, the structure around a soldier is both friend and hostile. It provides cover from both sight and weapons effects for both sides in the conflict. But the protection is not complete. Small openings can be created and used for visibility. Structures that appear to be solid may, in fact, have little or no protective value. Other structures, especially glass, may become lethal due to weapons effects.

None of this occurs in a quiet, clear air environment. Most building materials generate an enormous amount of dust when damaged. The resulting debris makes quiet movement nearly impossible. The soldier quickly finds himself in a noisy, cluttered space with zero visibility.

To make matters more complex, close infantry combat is extremely time sensitive. A difference in reaction time of a fraction of a second can determine the winner in a firefight. The only other form of combat that has time constants as small as close infantry combat is close air combat. While the logistic issues associated with air combat have created fiscally viable simulators, the logistics of infantry combat have demanded a much lower price point before they can be used effectively. However, as the performance/cost ratio of computers improves, infantry simulators become feasible.

Ultimately, the goal is to provide a virtual training environment that represents this environment with enough fidelity to provide a positive training experience without the expense and hazards associated with a live fire exercise.

5.6.1. Original Software Architecture

While a flight simulator may look at most of the world from altitudes greater than 10,000 feet, an infantry simulator must be able to provide detailed information on anything within the range of a sniper's rifle. During the proposal phase, this level of visual fidelity, combined with a requirement for low latency, forced a "heavy metal" computer solution. The only viable

computer platform was seen as high-end graphics platforms, such as Silicon Graphics (SGI) Onyx systems.

A single Onyx could provide excellent visuals to more than one user station, in a multiprocessor/multiprocessing system. The primary limitation, recognized from the beginning, was the cost of such a system. However, it was also recognized that technical advances would drive down the price point to a more acceptable level. Very few people however, recognized just how fast that point would be driven down! But given what was known at the time, combined with what support could be obtained for various vendors, the SGI, using the proprietary Performer software as a foundation, was the only viable choice for a platform. Support from Coryphaeus Software (now Centric Software) provided visual modeling tools.

These choices led to a number of design decisions. The hardware architecture of the platform was assumed to have multiple graphics heads on multi-processor single box configuration. Processors would be assigned to various tasks, with extra processors held in reserve for multi-threaded code. This configuration guaranteed the minimum latency with the reserve capacity to handle surges in number of active events, as would occur during a firefight.

The software architecture assumed a Performer™ base with shared memory as the primary form of data exchange. Given more than one processor in a system, Performer has built in extensions for multiprocessing. In a multi-processor system, the normal mode of operation for Performer is to form a pipeline, one processor to handle the main application, another processor culling geometry that is not contained within the view frustum, and a final processor rendering the geometry. It was also possible to spawn additional processes to handle other tasks associated with the application. As a matter of common terminology, a "thread" is a special case of a process that is designed to execute limited functionality in parallel with other threads. The main virtues of a thread are that it can be started up and shutdown more quickly than a full process, and that it has defined synchronization functions.

The language chosen for the application was C++, using classes to structure the underlying software. Three primary classes were defined, Space, Object, and Event. An "Event" is any interaction of Objects that can result in non-linear deformation of an Object. Opening a door is not an Event; triggering a satchel charge next to a door is an Event. Events are localized in space and time. An Event has an intrinsic time constant and statistical limit.

As an example, consider firing an M-16, full auto, inside a room. It has a nominal rate of fire of 600 rounds per minute, or 10 rounds per second. Since the basic unit of computer time is defined in video frames, this is one round every 3 to 12 frames depending on the current refresh rate (we will assume 60 Hz for the sake of discussion). So the first observation is that each round is fired in a different frame, each round from the M-16 triggers discrete Event. With a nominal muzzle velocity of approximately 3000 feet per second, a round will travel 50 feet per frame. The spatial limits of the Event are the Space that contains it for a maximum distance of 50 feet from the rounds previous location. The maximum time resolution required for computing the stepwise linear behavior of the round is on the order of 100 microseconds. This is a fine enough time step to resolve which of two rounds shattered a window, and hence, which one retained more velocity. The list of which Objects may be affected by the Event is limited to the current Space. This simplification is critical to bounding the complexity of the database. Other Spaces are only evaluated when the "active" Objects in an Event leave the limits of the current space (e.g., shoots through a wall, or passes through a portal).

The initial design of the Event process called for it to sleep (i.e., be inactive) where there were no Events to be processed. Its "class" determines the behavior of an Object and the action preformed on it. Therefore, a weapon will be "fired" if it is loaded, the safety is off, and the trigger is pulled. Various behaviors, such as firing a weapon, will initiate an Event. A pointer to the Object would be placed on the appropriate input queue of the Event process, and a signal would be used to "wake up" the process. The input queues are classed by the time constant of the potential event. Therefore, the detonating of a high explosive will be serviced sooner and more often than the flight of a .45 ACP round. Currently, there are 1 to 10 millisecond queues spaced in decades.

Internal to the Event process is a simple nested loop construct. When the process is awake, it processes all Events in its queues, servicing the fastest queues repeatedly until a slower queue is due for updating (e.g., a 1 microsecond queue is serviced 10 times before the 10 microsecond queue is serviced). Independent threads process all of the Events pending in a queue. If there are four free processors in the system, and six pending events in a queue, the first four events will be processed immediately and simultaneously, with the other two being processed as soon as two of the other Events complete their current cycle.

Since the processing time for given Event cycle is a function of the type of Event and the stage of the Event's "life", no assumptions are made to bound a particular thread. An example of a "fast" event cycle is a rifle bullet in flight. Baring a large amount of debris blown into its path, it will simply be stepped along its path toward the first Object that its trajectory intercepts. However, an explosion is a very computationally intensive Event cycle, depending on the assumptions applied. In the extreme case, the Hydrocodes could be running to accurately predict the effect of the blast or as currently envisioned pre-computed parametric solutions derived form analysis implemented external to the simulation.

Only at the end of the processing for an entire queue's cycle is synchronization imposed. The number of Events that can be handled in real time is a function of processor throughput and the number of available processors.

5.6.2 High Level Architecture (HLA) Requirements

HLA compliance has always been a requirement for this project. However, it was never considered a fundamental structure of the architecture. On the surface, the previous statement could cause concern in certain groups of the simulation community. But, in light of the objectives of this project, it makes perfect sense. Let us look at a simulation environment from an HLA perspective and consider its strengths and weakness.

An HLA compliant simulation environment consists of one or more Federates joined into a Federation. Internally, a Federate must be self-consistent. But across the Federation, there is no requirement for consistency among the Federates beyond the format and type of data exchanged between their Ambassadors and the Federations database.

As was mentioned above, a major concern from the outset of this project was communication latency between participants in a simulated urban combat environment. This latency takes two forms, the time required to format a message and the time to transfer the message. While very high bandwidth communications links help, the speed of light still takes its toll. Any routing

through servers only makes matters worse. Therefore, it is unlikely that significant training would occur between infantry participants located at remote sites. If the infantry participants are communicating through the Federation, the added latency will, most likely, make effective close combat training impossible. It should be noted that this constraint does not apply to non-real time analytical work. All of this implies that the results of this project will principally be applied within a single Federate.

However, the infantry does not fight a war alone. The mechanism must be provided to allow interaction with actual or simulated artillery and air units. Requests for fire support must be able to be passed out from the infantry Federate and the data for any incoming rounds passed back. At the same time, modification to the environment, with appropriate levels of detail, must also be passed out to update the visual simulation of other applicable Federates. An example of this problem is close air support from a helicopter unit. From the infantry unit's perspective, every bullet and the location of every person are of interest. From the air unit's perspective, only major explosions on the ground, rifle rounds actually hitting one of the helicopters and people outside are of interest.

Hence we have the conclusion that since the critical portion of this project resides within a single Federate, HLA compatibility does not drive the architecture of this system. However, the realistic requirement that this system be able to communicate with other simulation system demands that this system should be able to communicate with an HLA Federation and the STRICOM simulation architecture has been designed consistent with that objective.

5.6.3 Reasons for Revising Software Architecture

Three things have altered the map of simulation development over the last few years: cost, communications, and tools.

When this project started, SGI was the "only game in town." A suitable SGI platform was big and expensive, but that was what was required to get an adequate visual presentation. But the market for graphic oriented games on the PC has fueled a market for graphics cards that are nothing less than remarkable. This is not to say that PC based graphics systems are comparable to those of an SGI Infinite Reality system; they are not. But SGI has, for a number of reasons, slowed its development cycle on high performance systems, while new generations of PC graphics systems are coming out in as little as six months and being sold at consumer prices. "It's an SGI" is no longer adequate justification for the higher cost. Rather, requirements must be demonstrated for specific graphic features that are not currently available on PC systems. And with time, the difference in performance/features dwindles, while the price difference remains.

In the last two years a quiet revolution has occurred in computer communications. Commonly available bandwidth has gone up by over an order of magnitude, and practical bandwidth has gone up by more than two orders of magnitude. While only a short time ago, having a 10 Mbs Ethernet interface in a workstation was "state of the art", it is now reasonable to be using 100 Mbs in a home network. And with giga bit per second Ethernet, FCI, and IEEE 1394 interfaces available at any level, local network bandwidth is not much of an issue.

There is also the issue of availability of development tools. The cost of distribution for tools is negligible when compared to the cost of development. So a perspective developer will usually look for a system environment where the "acceptable price" times the number of units that can

be sold results in the greatest revenue. As the capabilities of the PC increased, there has been a significant market shift toward the PC. Most tools that were formally limited to high-end systems have migrated to the PC. As a case in point, SGI has announced a port of Performer to the Linux (the beta version is currently available for download from SGI). More significantly, there are many applications that are hosted on the PC only. At this point a note of caution is needed. The PC market is extremely volatile. Promised delivery dates are frequently pure fiction. Significant participants are subject to "overnight" changes in direction or status. So, until the equipment or software is delivered to your door, installed and tested, you can't count on it.

Then there is the issue of operating systems. Neither Unix (Irix on the SGI or Linux on the PC), nor any form of Windows was designed as a real time operating system. This is not to say that there aren't various work arounds and extensions, but the OS was not intended for the task to which it is being put. Irix has real time extensions. Linux, because of the access to the source code, can be modified. Windows is also not suitable. For this reason, the real time requirements of the target system must be realistically evaluated and prioritized before the selection of the run time environment is made.

Then the issue of operating system stability must be considered. Unix in general, and Irix for the purposes of our discussion, is as stable as it is currently possible to build a general-purpose operating system. Linux is evolving and stabilizing rapidly under intense scrutiny. Windows 98 is unsuitable for extensive simulation work, due to a lack of multi-processor support and a tendency to crash for no apparent reason (the loss of training time would quickly become unacceptable). Windows NT is more stable but slower than 98. It provides multi-processor support, but it is about to be superceded by Windows 2000. Windows 2000 is a relative unknown. Beta releases have been out for some time, but a lack of appropriate drives makes system evaluations difficult at best (Windows NT drivers will work, but are not optimized). For the current project, there are no hard requirements, so any of the operating systems is adequate. However, in an operational training system, where the quality of the training will have a direct affect on casualty rates, a more stringent evaluation must be made.

5.6.4 Current Software Configuration

The main structural difference in the software architecture is a shift from a unified, single box solution to a client/server model of operation. From the beginning, HLA compliance has required a degree of server functionality. Simulation Technologies, Inc (STI) was sub-contracted to provide the needed expertise in HLA. As the design progressed, and information and feedback was provided by STI, it became apparent that a more robust client/server model would be advantageous.

Then two other software considerations came into play. For a companion Phase II SBIR effort awarded to MRC by Soldier System Command (SSCOM) at Natick, a requirement for character animation was added, as either a virtual soldier, or as an avatar for a man-in-the-loop system. Given a build or buy decision, buy was the only reasonable solution. Given the available alternatives, Motivate™, from The Motion Factory,™ was selected. At the same time we were required to integrate the *MRCMAN* wound ballistics module from the MRC Soldier Protective Ensemble Computer Aided Design (SPE/CAD) system into the simulation as a first generation character simulator. Since both of these packages were hosted on Windows NT platforms, a strong client/server model became essential to the successful completion of both projects.

The entire decision process was iterative. The major factors were both immediate and long-term cost and performance. The only good news was the network bandwidth would not automatically preclude a client/server application. This is not to say that a sloppy design couldn't overload a reasonable network configuration. Rather, it was accepted that, for the cost of a degree of latency, a more general system configuration could be built. Examination of the trends in high-end SGI platforms versus the trends in the PC market resulted in one conclusion; in the short term, there is no perfect solution. But the PC has become a cost-effective platform for a number of applications. And the SGI remains the platform of choice when image quality is paramount.

These factors came together to reinforce the logic of switching to a client/server model. Given successful completion of the current STRICOM and SSCOM efforts, a number of potential applications have been identified. The most striking thing about these various applications is the lack of similarity among the operational requirements. Hence, the ability to "mix and match" the hardware platforms to meet the requirements of the target application becomes a significant asset.

The current system architecture (see Figure 18 below) uses a central database as a server. An integral part of this server is the Event processor. The decision to place the Event processor on the server is based in one requirement, maintain the lowest possible latency. Putting the Event processor anywhere else would increase the overall system latency by at least the delay of two additional communications links.

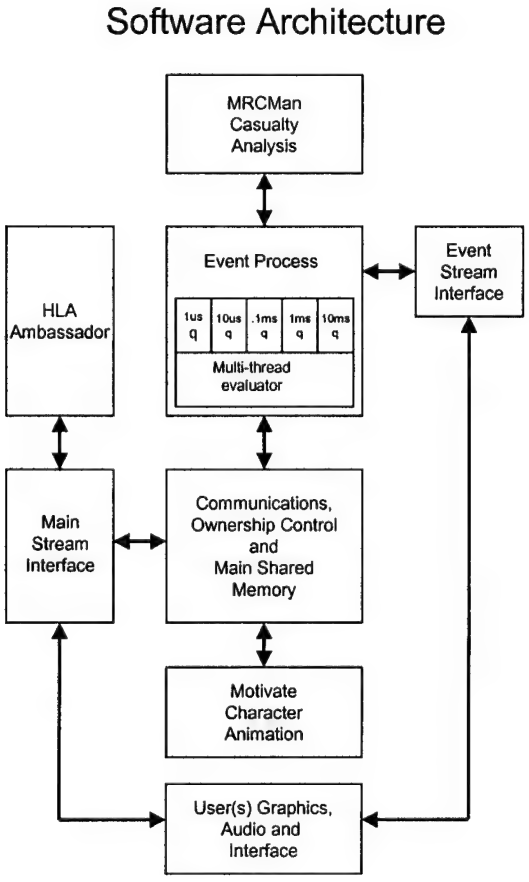


Figure 18. Current Software Configuration

There are no assumptions as to the actual communications mechanism between the server functions and the client functions. Communications can be broken down into three classes, data passed through local shared memory, data passed through distributed-shared memory (e.g., ScramNet), or data passed through a network protocol. In each case the arrival of data requires that the central server be notified of the arrival, so that the data redistributed to other servers and clients as needed. The architecture has to be transparent to the physical transport mechanism to allow for future extension. But for the near term, only a local shared memory and a stream interface was implemented.

This server is the ultimate "owner" of all data. However, temporary ownership of various data structures can be loaned to other server functions. One of the most immediate examples is a character animated by Motivate. The user inputs are sent to Motivate, resulting in changes to the character's state machine. This in turn causes a series of actions to be followed with time. At each frame, the Motivate interface outputs the current body position and joint rotations (see Figure 19). This data is passed to the server, which then distributes the data to the visualization system needing the data. However, if there were an event that applies an external force to the body, forcing it to move, the Event processor would take back partial or total control of the body in order to place it in the correct attitude. While it would allow a faster interface to have Motivate transfer body position data directly to the active clients, the need to have Event cleanly switch ownership makes this an unworkable option.

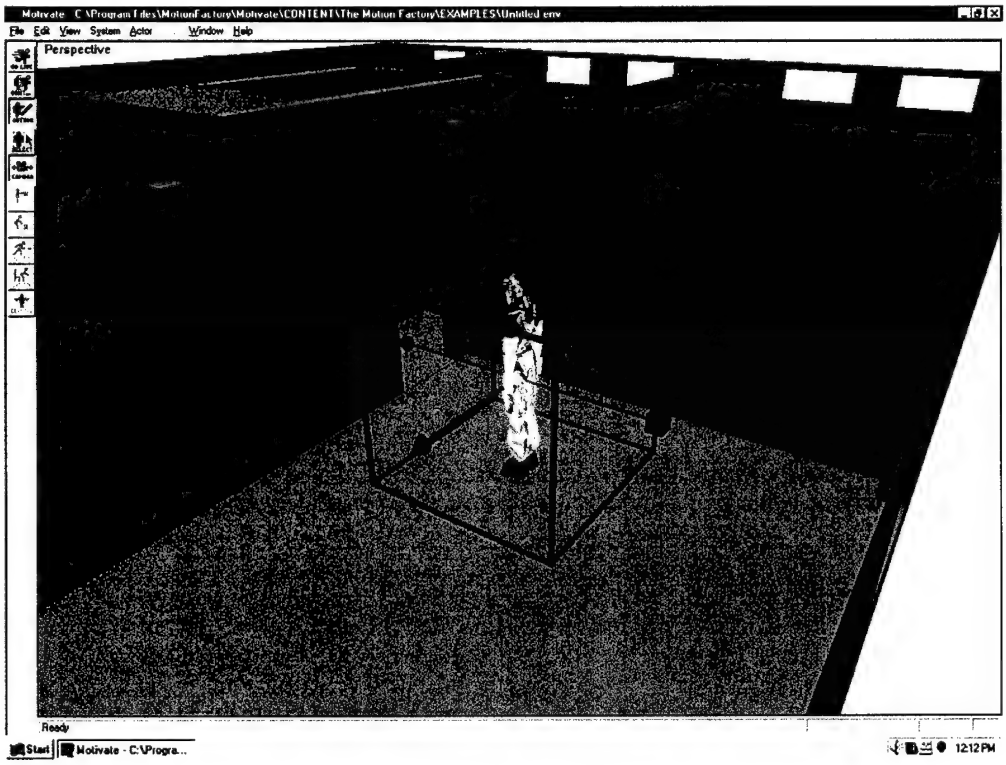


Figure 19. "APCLad" from the Motivate library being positioned on the second floor of the "town hall". The shot is from within Motivate. This was used to setup and test models before running them in the actual simulation.

If an Event impacts a human character in the simulation, the projectile type, velocity vector and impact point are passed to the Character Simulator (MRCMAN module) and the Event process. MRCMAN evaluates the effect of the impact and determines a casualty/non-casualty status of the person. The Event process determines any momentum transfer that would affect the stance of the person. This data is then passed through the central shared memory to Motivate. With the updated state, Motivate is then able to continue, or modify the behavior of the character (e.g., a running soldier hit in the torso by a rifle round, MRCMAN declares the soldier to be casualty, the Event process moves the body by some amount based on momentum transfer, and Motivate then has the character fall down).

All ballistic computations take place in the *Event Processor*. The Event Processor is designed to be multi-threaded to permit utilization of additional processors that may be present in the host system. Since multiple threads can be accessing the same body of code asynchronously, only static global values are used. All event specific code is passed in, or defined dynamically within the function.

There is one design decision that should be highlighted. During the conversion, several different approaches were investigated to maintain time coherency in the simulation. Each technique had advantages and disadvantages, depending on the end application. The methodology, discussed below, was decided upon based on a "reasonable" level of time and spatial coherency in exchange for "reasonable" computational efficiency.

The sequence of operations is based on the assumptions that a simulated round is fired in an urban environment and that the fidelity of the simulation is driven by the real time performance of the graphics subsystem. The urban environment implies that the mean free path of any projectile is short compared to the theoretical range of modern weapons. The interest in graphics performance defines the smallest practical granularity of time to be one frame period. Everything that happens within a single frame period appears to be simultaneous; hence computing to finer time resolution becomes pointless to the end presentation.

An Event is initiated by firing the weapon. This result in a projectile being introduced into the environment at the location defined by the muzzle of the rifle, moving with a velocity vector defined by the attitude of the muzzle and the internal ballistics of the round and the weapon. While the position and velocity are subject to statistical variation based on a number of factors, it is assumed for the immediate discussion that the position and velocity of the round is known.

Given the muzzle position and the muzzle velocity, a straight-line segment can be "drawn" to approximate the trajectory of the round during the first frame period (between 10 and 17 ms, depending on the frame rate). This line segment is passed to the Performer collision detection algorithm and it is tested against the objects, and the bounding volumes of any sub-spaces, listed as being in the current space. Either the round will hit something in the current frame period, or it will not. If the round doesn't hit anything, a non-linear trajectory is computed to determine the position and velocity vector of the round at the end of the frame period. This revised data is used as the initial condition for the processing of the round's trajectory during the next frame.

If the round does "hit" something, the first/nearest thing hit is evaluated. If the object was a portal, the round is leaving the current space and is checked for collisions with objects defined to be members of the space on the other side of the portal. If the "object" intersected is actually the bounding volume of a sub-space, the objects in that sub-space is also be tested for collision. If an

actual object is hit, the approximate impact location is noted and a linear approximation for the round's trajectory is computed. This is used to determine the actual impact point.

At this point, we reach one of the limiting factors in the simulation. The limitation is that it is designed with real-time simulation as a design priority. For a strictly analytical application, the statistical distribution applied to most initial values, combined with the granularity of time could limit the validity of the results. However, the time granularity can be an explicitly set value. Rather than relying on the vertical refresh rate to determine the time granularity, an arbitrary value, possibly down to the microsecond level if needed, can be set. It is also possible to replace the functions or disable the statistical perturbations that are applied to various values.

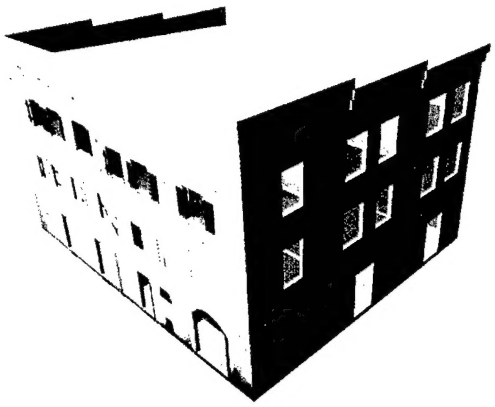
Given that a projectile has impacted a surface, the results of that impact must now be determined. Based on ballistic tests that were performed during this effort, the terminal behavior of a projectile is represented in the ballistics code by two parameters, V_{c1} and V_{c2}. These values are critical speeds of the projectile with respect to the material of the surface impacted along the normal. If the speed of impact is below V_{c1}, the round will not rebound/ricochet off the target. And, as noted in Table 6, the ricochet will depart at a low angle, nearly parallel to the impacted surface.

If the round impacts with a speed greater than V_{c2}, the round will penetrate the target, retaining any surplus energy. As can be observed from the data in Table 6, several commonly used rounds could pass entirely through most common buildings, and still maintain a lethally high velocity. If the speed at impact is between V_{c1} and V_{c2}, the round will become embedded in the impacted surface.

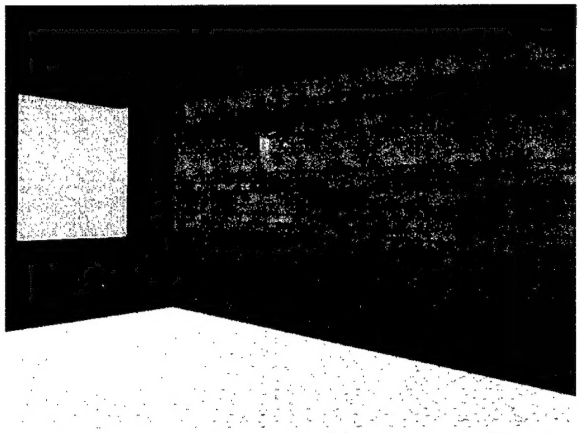
The Section 5.0 Appendix shows selected images from visual simulations emphasizing bullet holes from small arms.

SECTION 5.0 APPENDIX

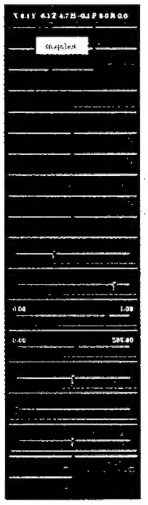
SELECTED IMAGES FROM VISUALIZATION SIMULATION



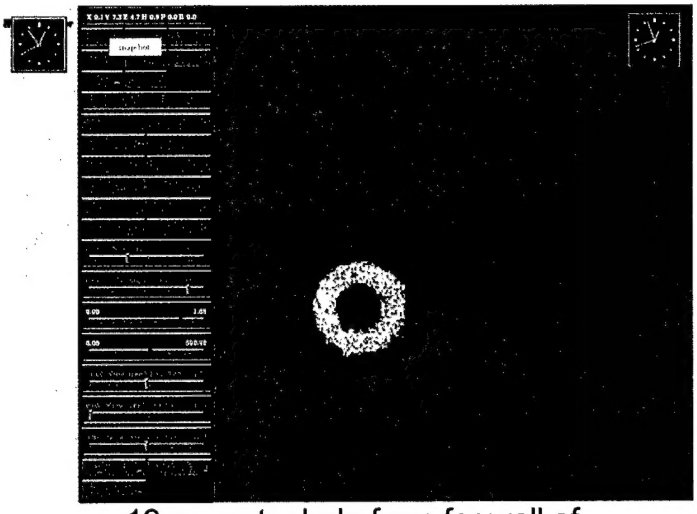
McKenna Town Hall



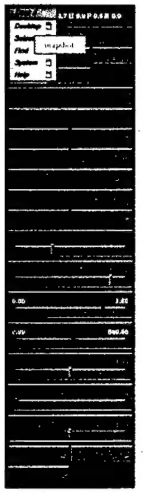
Room on Second Floor with Shot Gun Hole in Wall



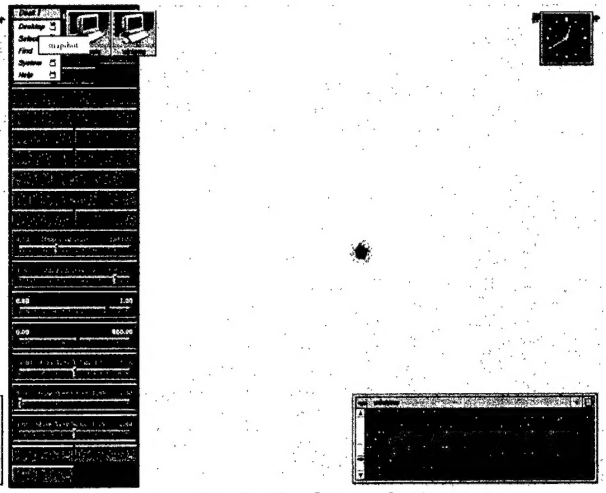
12 ga. Shot Gun slug entering building



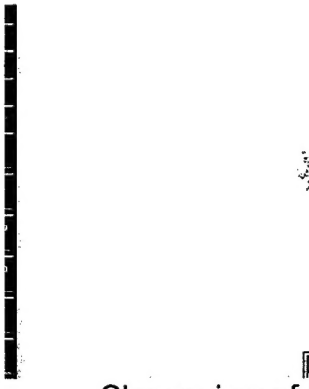
12 ga. entry hole from far wall of building (having passed through four interior walls) before exiting building (note that the height of hole is less than entry hole when round first entered building)



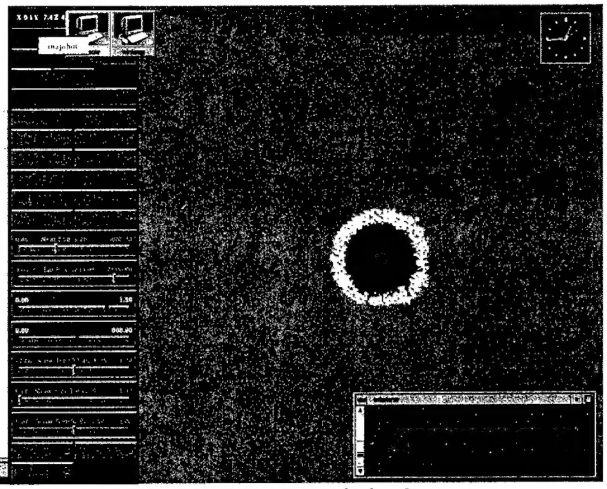
12 ga. slug entry hole from 0.5 meters



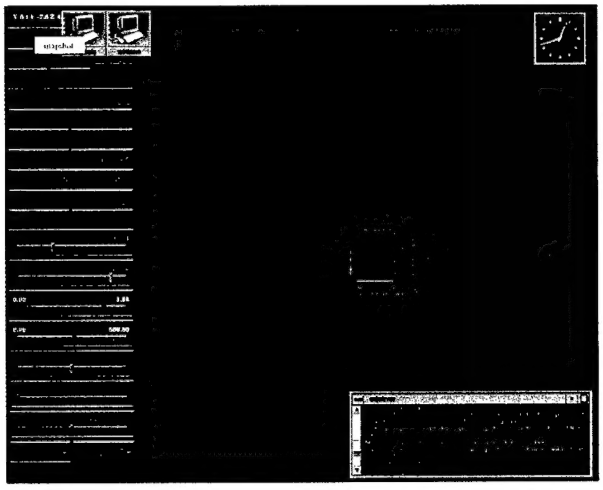
5.56-mm entry hole from 0.5 meters



Closer view of 5.56 entry hole



5.56-mm exit hole
(note exit hole larger than entry hole



Inside of 5.56-mm hole looking back through hole

SECTION 6.0

CONCLUSIONS, RECOMMENDATIONS, AND FUTURE EXTENSIONS

The goal of this project has always been to develop a real-time simulation of weapon effects in a 3D urban warfare virtual environment. The second floor of the Town Hall at the McKenna MOUT site at Fort Benning was selected as the demonstration environment. The geometry of this building was configured into a three-dimensional digital database. Sub-real-time models describing the physics of various small arms interacting with this environment were developed. Mechanical and structural properties corresponding to various building constructions were mapped onto the visual database so that the physics modules could read the data and describe interaction with various small arms that might be used in a building clearing operation.

All of the building elements that are included with the simulation were built and tested against various ballistic threats to validate and calibrate the physics models that were developed. Blast models were correlated against empirical data from various reports and by agreement with the Army Waterways Experimental Station Blast-X code.

A fist generation character simulator interfaces with the simulation. This character simulator was developed in another MRC SBIR and assesses the effect of penetrating wounds. The character simulator was animated using the Motivate™ software package that executes on an NT server. The character simulator interface was developed so that when a more advanced character simulator, currently under development in a parallel Phase II SBIR funded by the U.S. Army Soldier System Commend (SSCOM), is available it can be easily "swapped" with the first generation character simulator. This more advanced character simulator handles blast, blunt trauma, and tertiary blast injuries due to collision with secondary objects. The anatomical model in this second-generation character simulator is also based on the National Library of Medicine's Visible Human data set. In this more powerful second-generation character simulator the Visible Human anatomical data can be scaled to fit the contours of laser scanned people participating in the simulation.

Future work related to this project can be resolved in to three areas. These areas concern: (1) generalizing and automating the techniques so that the weapon effects can be imported to other simulations, (2) extensions to the structural process simulator, and (3) extensions to the character process simulator.

Future work related to generalizing the techniques developed include automating the parsing of the visual and mechanical/structural database, making the software platform independent and in particular porting the software to a Linux operating system with Performer support.

Structural process extensions include extending the second floor simulation on the McKenna Town Hall to the entire McKenna MOUT site. MRC already has high-resolution data files for this site. Other extensions include incorporating other building constructions, other weapon types

(including non-lethal weapons or for example, diffusion of toxic gases buildings) and higher fidelity weapon effects models.

Many simplifying assumptions were made in the physical interaction of the weapon effect with the environment. For example, when a bullet strikes room-furniture, e.g., a chair or a desk, the bullet may penetrate through the entry location, chips may eject, a chair or desk may suffer both translational and rotational motion or the bullet may ricochet after impact. In the current modeling effort, these effects have been included to various orders of accuracy. The chipping mechanisms have not been investigated fully. The ejection velocity (both magnitude and direction) as well as the mass, shape and sizes of the chips have not been modeled to a high level of detail. One reason is that the resulting physical problem is not deterministic due to the randomness in the associated physical properties. This necessitates a statistical modeling of the problem that, in turn, requires a large amount of experimental data from bullet impact tests on furniture.

Another area in which the fidelity of the structural process modeling could be improved is the motion of furniture due to bullet impact or blast loading. It has been pointed out in the sections describing the motion of desk and chair that the resulting problem is nonlinear due to the coupling of the two modes of motion (translation and rotation). Floor friction is one of the reasons for such nonlinear coupling. Thus the temporal history of rotation cannot be calculated independent of the temporal history of the translational motion of the chair or desk. For determining the first order effect due to bullet impact or blast loading, the equations of motion were decoupled. Future work could include the solution of the full nonlinear equations of motion that yield more realistic motion of room furniture.

Character process extensions include incorporating more behaviors; for example, different movement strategies, more varied response to injury (currently, the response is bi-modal – operational casualty or not), and other forms of trauma (e.g., heat stress). Also, new types of equipment could be added, for example sensors, protective armor, etc.